AEROSPACE AND DEFENSE
MAY 2025
ISSUE 418

circuitcellar.com

circuit cellar

Inspiring the Evolution of Embedded Design

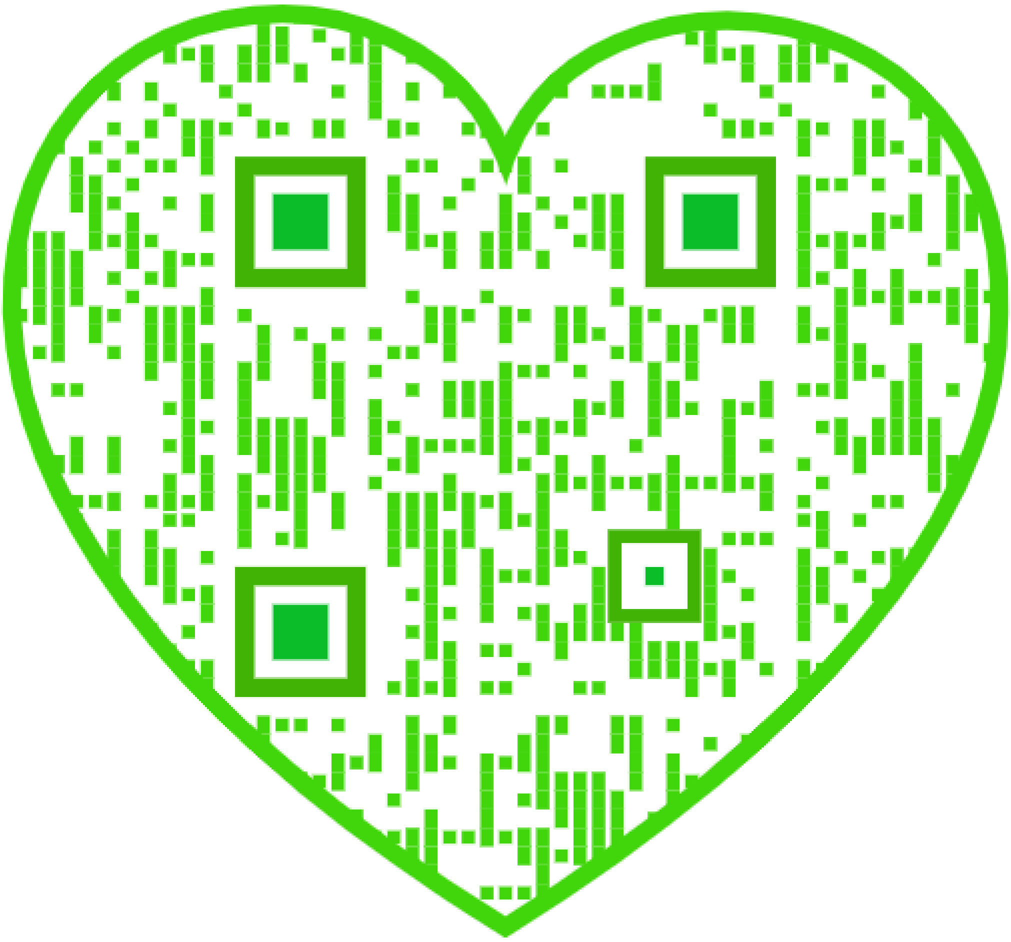
AEROSPACE AND DEFENSE

IoT Boards and Modules | OBD2 Protocol Automotive Telemetry System | Extending Battery Power with Wake-Up Receivers | 3D-Printed Robotic Actuator | ESP32 MCU 3-Phase Energy Meter

Experimenting with CHERI on the Sonata Board | Implementing Matter Protocol

The Future of Quantum Computing Security

$9.00US $10.00CAN
05>
0 74470 75349 0



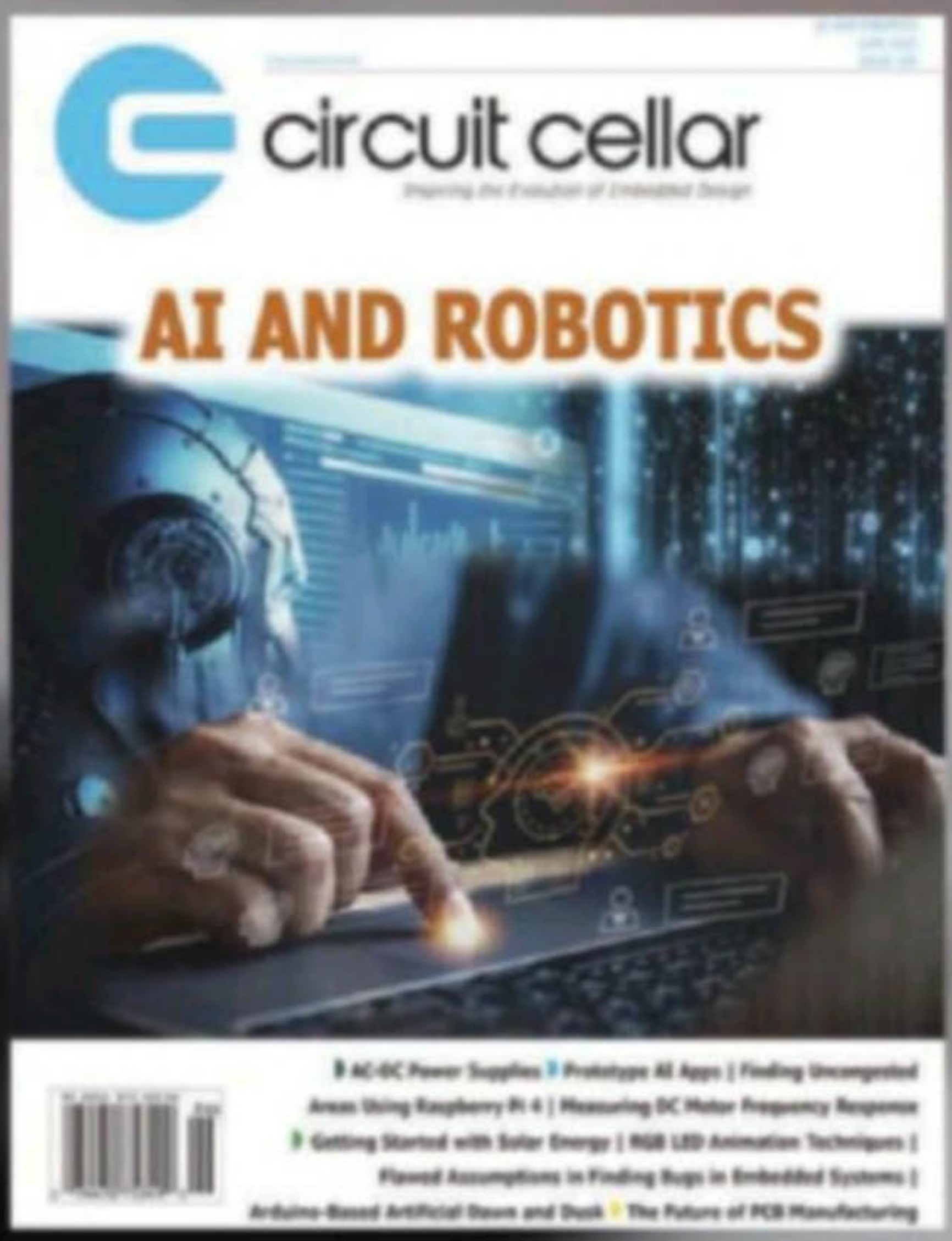
Imagine what you will
Discover
circuit cellar VAULT
CC VAULT
circuit cellar
SMART BUILDINGS AND CITIES
circuit cellar
DATA ACQUISITION TECHNOLOGY
circuit cellar
PCB DESIGN & PROTOTYPING
circuit cellar
TINY EMBEDDED BOARDS

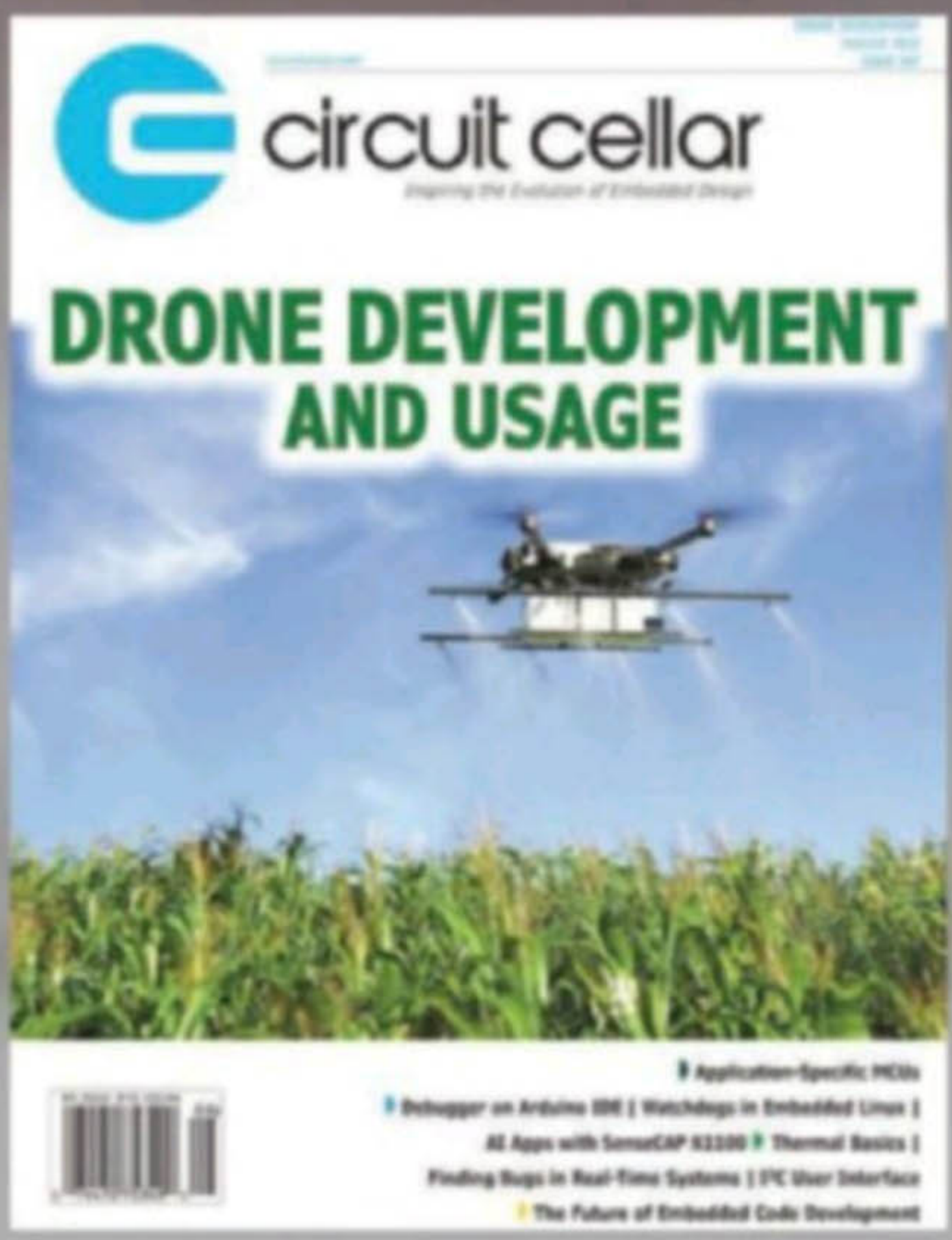
400+ ISSUES

Get the complete *Circuit Cellar* issue archive and article code stored on a stylish, durable and portable USB flash drive. You can easily keep your CC Vault archive up to date by purchasing subsequent issues from our webshop or by downloading issues with a *Circuit Cellar* Digital Subscription. Issues appear in searchable PDF format.

cc-webshop.com

circuit cellar
WEARABLE DEVICES
IoT Boards and Modules
Device Trees in Embedded Linux | Understanding Ground Stitching Vias | AWS for IoT Projects
Fault Injection and Power Analysis for Appliance Repair | A Lock-In Amplifier Based on a PSoC5LP SoC | Build an ESP32-to-Raspberry Pi Bridge
The Future of Experience Driving


circuit cellar
AI AND ROBOTICS
AC-DC Power Supplies
Prototype AI Apps | Finding Uncongested Areas Using Raspberry Pi 4 | Measuring DC Motor Frequency Response
Getting Started with Solar Energy | RGB LED Animation Techniques | Flawed Assumptions in Finding Bugs in Embedded Systems | Arduino-Based Artificial Dawn and Dusk
The Future of PCB Manufacturing


circuit cellar
DRONE DEVELOPMENT AND USAGE
Application-Specific MCUs
Debugger on Arduino IDE | Watchdogs in Embedded Linux | AI Apps with SenseCAP K1100
Thermal Basics | Finding Bugs in Real-Time Systems | I²C User Interface
The Future of Embedded Code Development

circuit cellar
MCU DEVELOPMENT TOOLS
Power/Battery Management ICs
PIC16 Controller for Cifra 5 Flip Clock | Raspberry Pi Zero Lego Robotics | Music Transcriber via Pi | Versatile Assistive Tech
Discover InPlay's NanoBeacon | Low Voltage Solar Energy System
The Future of Design Flexibility

Complete archive includes PDFs of all issues in print through date of purchase.

INPUT Voltage

From Fantasy to Fact

This month's tech feature embarks on a journey through the evolution of aerospace and defense technology, contrasting the visionary fantasies of earlier decades with the realities of today. Using the iconic 1977 film *Star Wars* as a benchmark against which to measure modern advancements. I examine communications software, missile guidance systems, and integrated radar detection systems and compare their efficacy to that of the technology seen in the film. It's remarkable progress given the complexities and nuances of real-world technological innovation verses the imagination of science fiction. Can't help but wonder if the film inspired the innovation, or what led to the creative advancements often presented in *Circuit Cellar* articles.

An article by Abel Didouh uses wake-up receivers to conserve power in battery operated devices. In his article "Wake-Up Receivers: Extending Operational Periods in Battery-Powered Applications" Didouh showcases the cutting edge of wake-up receivers for IoT applications as a means to conserve power, and then goes on to discuss the integration of specific use cases.

For an interesting at-home project, Gamal Labib outlines their project to build a 3-phase energy meter in their article "A 3-Phase Energy Meter with Multicore Support." Labib constructs the device using an ESP 32 microprocessor with voltage and current sensors to calculate and display energy consumption, and store the collected data to an SD card. Labib guides you through the hardware setup and programming of the device in a comprehensive and educational article.

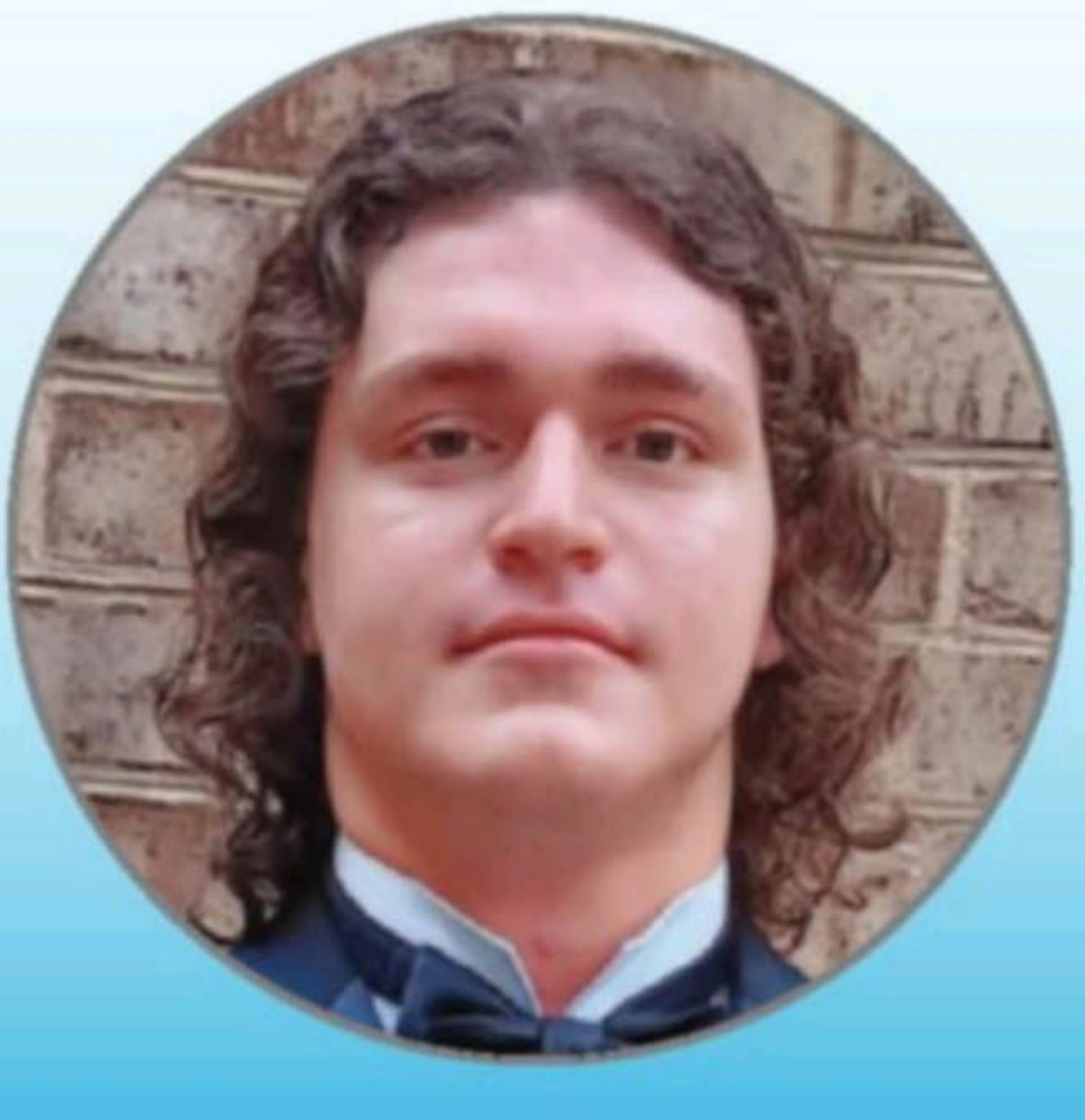
In a joint effort Maheshwaran Sadasivam and Vairamani Kanagavel outline the design and at-home construction of a robotic actuator in "Design and Implementation of a 3D-Printed Robotic Actuator." The design uses a simple, yet educational project for students seeking to become involved in robotics. For a budget of approximately $500, Sadasivam and Kanagavel give thorough instructions for an encoded stepper motor and 3D printed planetary gearbox to create a stepper motor controlled via the CAN protocol.

Alex Moore, Illia Pavych, Matthew Pearson, and Adam Schmidt, students from Camosun College, showcase their capstone project in "LapLogic Wireless Automotive Telemetry System" where they use a real-time OBD2 scanner connected to the LapLogic Hub app to wirelessly receive real-time information on traditional and obscure engine data. Follow along on their three-month journey as they prototype, build, code, and test all aspects of this project.

In "Implementing the Matter Protocol" Brian Millier follows up on previous articles regarding the use of Amazon Alexa. Millier outlines the features and uses of the Arduino Nano Matter board and Sparkfun's Thing Plus board. As part of the Early Adopters program for Arduino, Millier gives first of its kind commentary on the functionality and capability of this new device, as he uses it to make and control various IoT devices from his Alexa smart speaker.

In "Experimenting With CHERI on the Sonata Board" Colin O'Flynn experiments with the CHERI extension for RISC-V using the CHERIoT architecture on the open source Sonata Board to create fundamentally secure systems running on C and C++ programming, without the need to create a soft-core processor. As a follow up to an article by O'Flynn in 2023, this article finally delivers an easy hardware solution to experiment with this exciting program.

Regardless of the source of inspiration technology developments continue to take fantasy to fact. I for one, look forward to seeing some of my favorite sci-fi movie tech in the real world.



Caleb Smith

Editor-In-Chief
csmith@circuitcellar.com

Issue 418 May 2025 | ISSN 1528-0608

CIRCUIT CELLAR® (ISSN 1528-0608) is published monthly by:

KCK Media Corp.
PO Box 417, Chase City, VA 23924

Periodical rates paid at Chase City, VA, and additional offices. One-year (12 issues) subscription rate US and possessions $50, Canada $65, Foreign/ ROW $75. All subscription orders payable in US funds only via Visa, MasterCard, international postal money order, or check drawn on US bank.

SUBSCRIPTION MANAGEMENT

Online Account Management: circuitcellar.com/account
Renew | Change Address/E-mail | Check Status

CUSTOMER SERVICE

E-mail: customerservice@circuitcellar.com

Phone: 434.533.0246

Mail: Circuit Cellar, PO Box 417, Chase City, VA 23924

Postmaster: Send address changes to
Circuit Cellar, PO Box 417, Chase City, VA 23924

NEW SUBSCRIPTIONS

circuitcellar.com/subscription

ADVERTISING

Contact: Hugh Heinsohn

Phone: 757-525-3677

Fax: 888-980-1303

E-mail: hheinsohn@circuitcellar.com
Advertising rates and terms available on request.

NEW PRODUCTS

E-mail: product-editor@circuitcellar.com

HEAD OFFICE

KCK Media Corp.
PO Box 417
Chase City, VA 23924
Phone: 434-533-0246

COPYRIGHT NOTICE

Entire contents copyright © 2025 by KCK Media Corp. All rights reserved. Circuit Cellar is a registered trademark of KCK Media Corp. Reproduction of this publication in whole or in part without written consent from KCK Media Corp. is prohibited.

DISCLAIMER

KCK Media Corp. makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors printed in Circuit Cellar®. Furthermore, because of possible variation in the quality and condition of materials and workmanship of reader-assembled projects, KCK Media Corp. disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published in Circuit Cellar®.

The information provided in Circuit Cellar® by KCK Media Corp. is for educational purposes. KCK Media Corp. makes no claims or warrants that readers have a right to build things based upon these ideas under patent or other relevant intellectual property law in their jurisdiction, or that readers have a right to construct or operate any of the devices described herein under the relevant patent or other intellectual property law of the reader's jurisdiction. The reader assumes any risk of infringement liability for constructing or operating such devices.

© KCK Media Corp. 2025 Printed in the United States

OUR NETWORK

circuit cellar
LinuxGizmos
audioxpress
VOICE COIL
LIS LOUDSPEAKER INDUSTRY SOURCEBOOK

SUPPORTING COMPANIES

Beningo Embedded Group	47
Blackmagic Design	11
BusBoard Prototype Systems	15
CCS, Inc.	61
EmbeddedTS	61, C4
Recom	60
SeaFire Micros, Inc.	C3
X-ES	25

NOT A SUPPORTING COMPANY YET?

Contact Hugh Heinsohn
hugh@circuitcellar.com,
Phone: 757-525-3677,
Fax: 888-980-1303
to reserve space in the
next issue of *Circuit Cellar.*

THE TEAM

FOUNDER Steve Ciarcia
PUBLISHER KC Prescott
CONTROLLER Chuck Fellows
EDITOR-IN-CHIEF Caleb Smith
CONTRIBUTING EDITOR Brian Millier
CONTENT MANAGER- EU Jurij Mikeln
SENIOR ASSOCIATE EDITOR Shannon Becker
TECHNICAL COPY EDITOR Carol Bower
PROJECT EDITORS Ken Davidson
David Tweed
MARKETING MANAGER Tori Zienka
ADVERTISING COORDINATOR
Diana Winsor
ADVERTISING SALES REP.
Hugh Heinsohn

COLUMNISTS
Stuart Ball (Start to Finish)
Jacob Beningo (Modern Embedded Insights)
Bob Japenga (Embedded in Thin Slices)
Brian Millier (Picking Up Mixed Signals)
Colin O'Flynn (Embedded Systems Essentials)

FEATURES

4 **LapLogic Wireless Automotive Telemetry System**
Using Onboard Diagnostic 2 (OBD2) Protocol
By Alex Moore, Illia Pavych, Matthew Pearson, and Adam Schmidt

12 **Wake-Up Receivers**
Extending Operational Periods in Battery-Powered Applications
By Abel Didouh

18 **Design and Implementation of a 3D-Printed Robotic Actuator**
By Maheshwaran Sadasivam and Vairamani Kanagavel

26 **A 3-Phase Energy Meter with Multicore Support**
Using an ESP32 MCU with Voltage and Current Sensors
By Gamal Labib

33 TECHNOLOGY FEATURE
Aerospace and Defense
Outshining Science Fiction
By Caleb Smith

38 DATASHEET
IoT Boards and Modules
A New Era of Connectivity and Intelligence
By Caleb Smith

COLUMNS

42 Embedded System Essentials
Experimenting with CHERI on the Sonata Board
By Colin O'Flynn

48 Picking Up Mixed Signals
Implementing the Matter Protocol
Using the Arduino Nano Matter or Sparkfun Thing Plus Matter Boards
By Brian Millier

62 TECH THE FUTURE
The Future of Cryptography
PQCryptoLib-Embedded and the Quantum Threat
By Michele Sartori

58 : PRODUCT NEWS
60 : TEST YOUR EQ

@editor_cc
@circuitcellar
circuitcellar
CircuitCellar

LapLogic Wireless Automotive Telemetry System

Using Onboard Diagnostic 2 (OBD2) Protocol

By
Alex Moore, Illia Pavych,
Matthew Pearson, and Adam Schmidt

Explore how a team of Camosun College students developed a scanner that uses the OBD2 protocol to monitor a car's inner workings, and delivers real-time vehicle data directly to an app on a smartphone, with both mobile and remote functionalities. This innovative project enhances driving experiences and provides valuable insights for car enthusiasts and racers, alike.

Imagine driving on a racetrack, pushing your car to its limits, and wanting detailed insights beyond basic metrics of speed or engine RPM. What about less obvious stats such as engine load, fuel efficiency, or intake air temperature? What if you could access this hidden data—and more—in real time, right from your smartphone?

This vision drove our team of final-semester Camosun College students to develop a wireless automotive telemetry system (WATS) as our capstone project. Over the course of our program, we learned about a variety of topics—from microcontrollers and sensors to communication protocols and mobile app development. Our project allowed us to combine these skills and create something practical and exciting.

Our system comprises the WATS scanner and a smartphone application called the LapLogic Hub. Unlike traditional vehicle dashboards that display common statistics, our scanner taps into a car's sensors to provide both conventional metrics (such as engine RPM and speed) and lesser-known performance data, such as calculated engine load and intake air temperature, using the On-board Diagnostic 2 (OBD2) protocol. With information transmitted wirelessly from our scanner to our mobile app, users gain comprehensive insights into their car's performance, whether for racing, tuning, or personal curiosity.

The app offers two modes of data access: CAR mode, which connects via Bluetooth for real-time data while driving; and REMOTE mode, which allows users to monitor vehicle performance in real-time from anywhere in the world. This seamless integration within a single application makes it versatile for both on-track performance monitoring and off-site analysis.

Throughout the project, we faced several challenges, including establishing reliable OBD2 communication, setting up Bluetooth connectivity, and creating a seamless app experience using .NET MAUI, a cross-platform framework for building desktop and mobile apps. We also leveraged our skills in printed circuit board (PCB) design and 3D printing to build the hardware needed for the system. The result is a scanner that transforms how car data is accessed, allowing users to explore both familiar and unconventional metrics from their vehicles. The finished scanner that plugs into the OBD2 port of a car is shown in **Figure 1**, and **Figure 2** provides a visual representation of how our system works.

COMMUNICATING WITH A VEHICLE

To establish communication with a vehicle, it was essential to connect to its internal communication system. To make this connection, we had to understand two critical components: the type of communication system the vehicle used, known as the Controller Area Network (CAN) bus; and the standardized language for accessing this system, the OBD2 protocol.

CAN Bus: The CAN bus serves as the backbone of a vehicle's internal communication, functioning much like the human nervous system. It allows various control modules, such as the engine control unit (ECU) or transmission control unit (TCU), to exchange data efficiently and reliably. While older vehicles may use alternative communication systems, the CAN bus has become the industry standard, due to its stability, robustness against electrical noise, and scalability.

Given its widespread adoption in modern vehicles and its role as the future of automotive communication hardware, we designed our project to be compatible specifically with the CAN bus. This decision ensures compatibility with most cars on the road today.

OBD2 Protocol: While the CAN bus facilitates communication among a vehicle's internal systems, external devices like ours need a standardized protocol to access and interpret this communication. This is where the OBD2 protocol comes into play.

The OBD2 protocol acts as a universal language that manufacturers adhere to, enabling consistent and reliable access to a vehicle's diagnostic and operational data. It defines how requests for information are sent to the vehicle, how the data is structured, and how responses are formatted. By utilizing the OBD2 protocol, we can query the vehicle for information such as engine speed, vehicle speed, and diagnostic trouble codes, all while ensuring compatibility across manufacturers and models. A more detailed explanation of how information is requested from the vehicle can be found in the "Firmware" section of this article.

HARDWARE

We selected each hardware component in our scanner to ensure seamless integration, fulfill design requirements, and maintain compatibility with automotive systems such as the CAN bus and OBD2 protocol. Below, we outline the major hardware elements and the rationale behind their selection. The microcontroller is on one side of a PCB and the CAN transceiver and buck converter are on the other side, as detailed below.

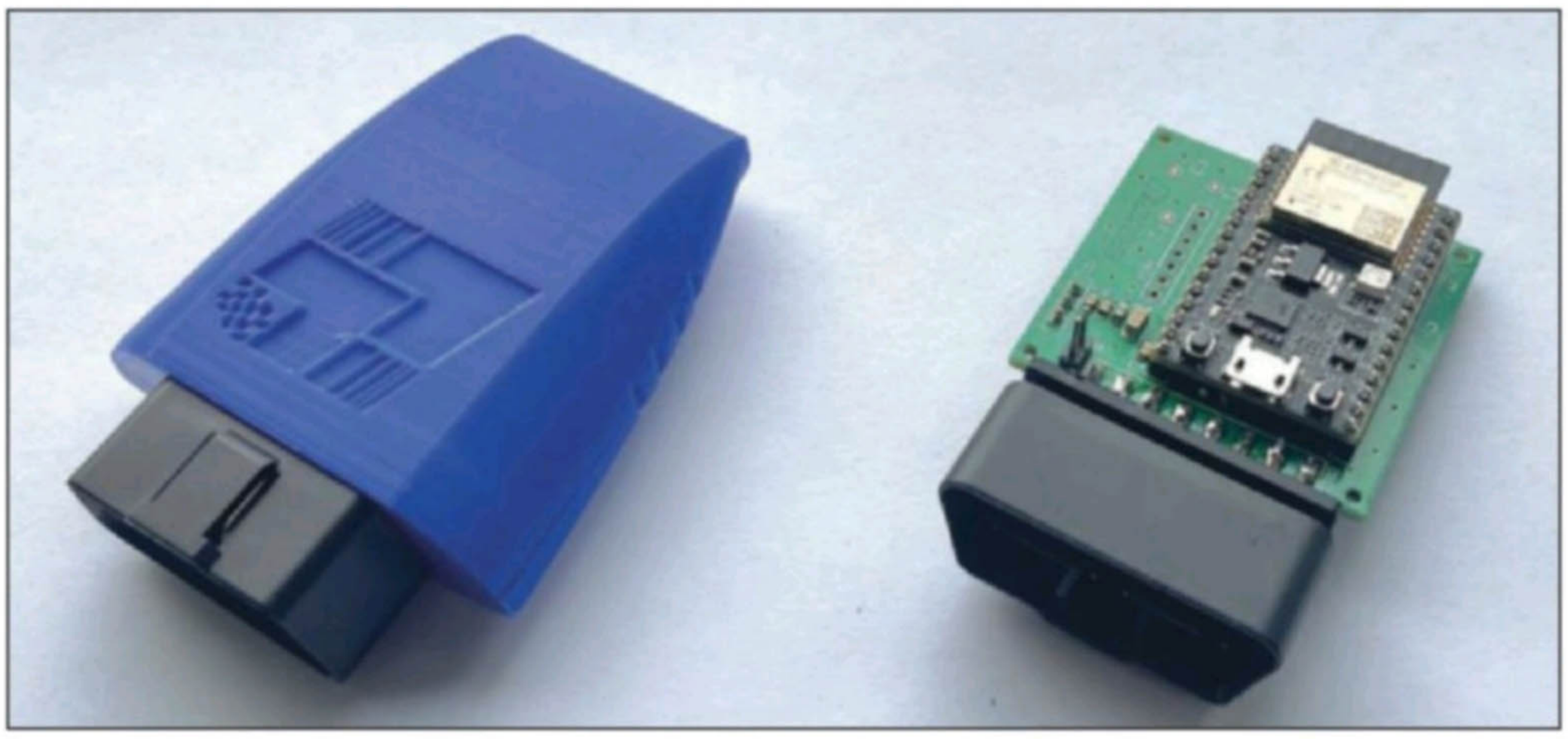


FIGURE 1
Views of the LapLogic scanner that plugs into the car's OBD2. The one on the left is encased in a 3D-printed enclosure. The one on the right has no enclosure to showcase the inner workings.

ESP32-C3 Microcontroller: To interface with the automotive CAN Bus and enable Bluetooth communication, we chose the ESP32-C3-devkit-02 (**Figure 3**). This microcontroller's Bluetooth Low-Energy (BLE) and Two-Wire Automotive Interface (TWAI) capabilities make it an ideal choice for our system. It serves as the central hub, sending OBD2 requests to the vehicle and wirelessly relaying the data to the mobile application.

TJA1050 CAN Transceiver: Automotive systems use the CAN protocol, whereas the ESP32 communicates using TWAI. To bridge this gap, we selected the TJA1050 CAN Transceiver (Figure 3). This compact and inexpensive ($2.99) component translates the microprocessor's requests into CAN-compatible messages. Its simple setup requires only a 5V power supply and direct connections to the ESP32's TWAI pins, facilitating seamless communication between the microcontroller and the vehicle's CAN Bus.

TPSM84205 Switching Regulator: To power our scanner, we utilized the 12V supply from the vehicle's OBD2 port. The Texas Instruments TPSM84205 switching buck

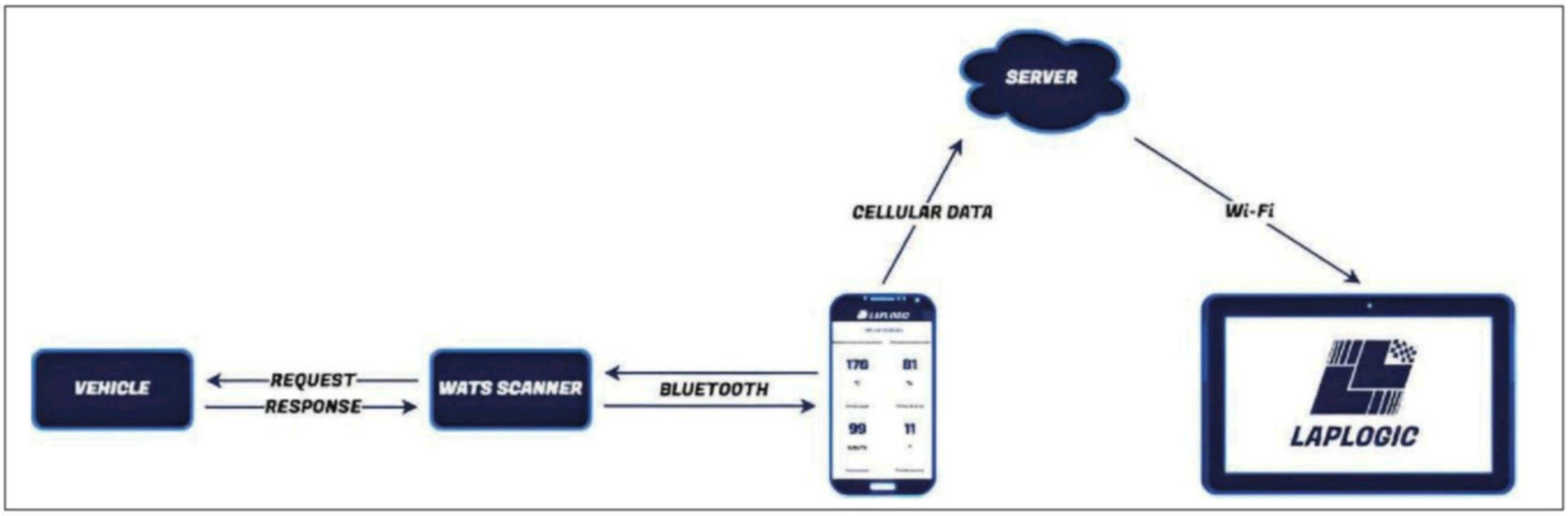


FIGURE 2
Flow chart of the whole system, illustrating how data flows through the various system components. The smartphone represents the CAR mode of our application, and the tablet represents the REMOTE mode.

converter (Figure 3) steps down the voltage to 5V, providing the required power for our scanner. This regulator tolerates a wide range of input voltages (7V to 28V), accommodating fluctuations in a car's electrical system during startup or under other conditions, such as cold weather. Its small size and wide input voltage range made it the practical choice for ensuring stable power delivery in our scanner.

PCB Design: Once we finalized the components, we moved to the PCB design phase to efficiently integrate them into a cohesive system. Our main objectives were minimizing the size of the PCB and reducing electrical noise to ensure compatibility with the sensitive automotive CAN bus.

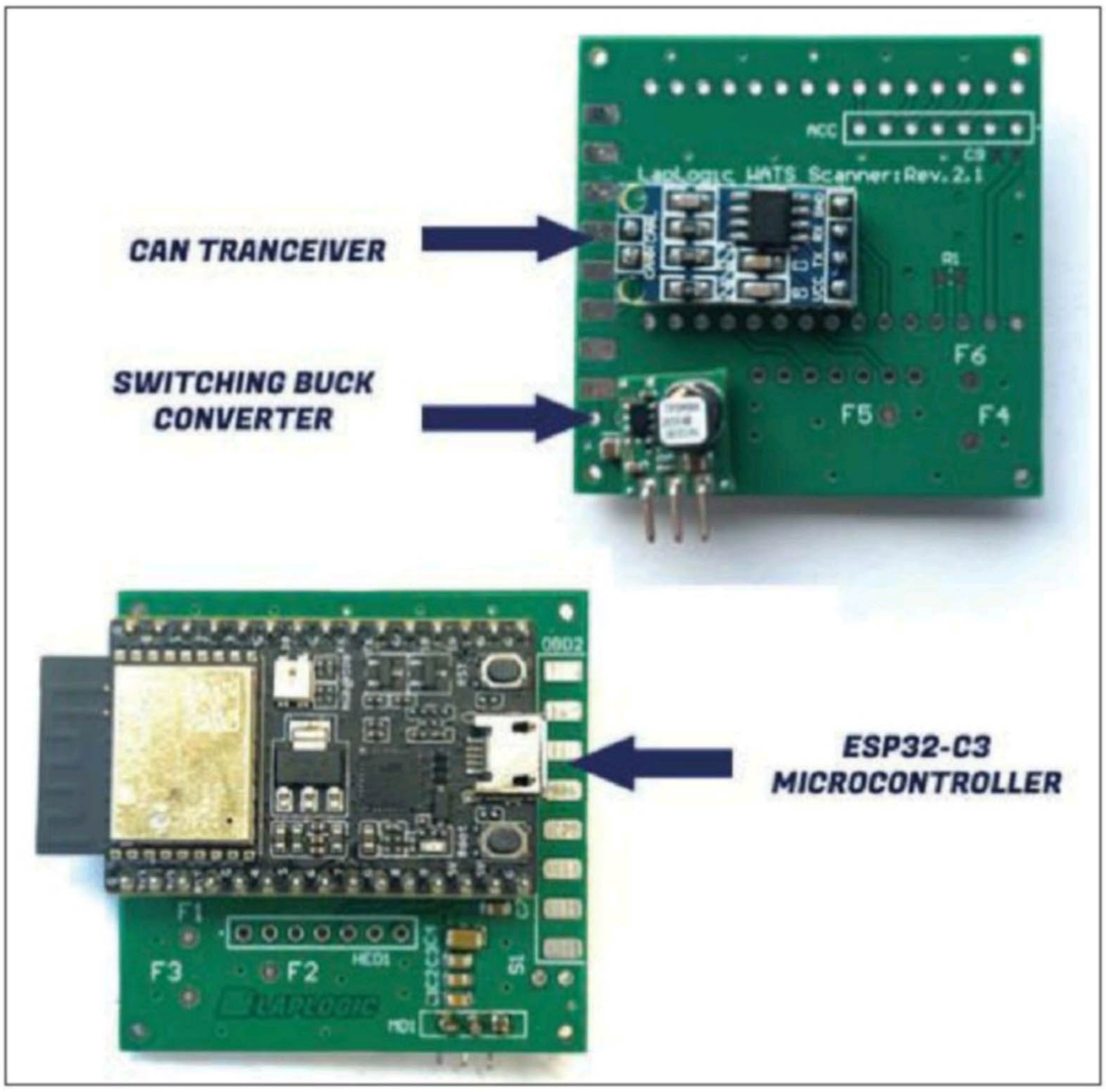


FIGURE 3
Both sides of the PCB, showcasing selected components: the ESP32-C3 microcontroller, the Texas Instruments switching buck converter, and the TJA1050 CAN transceiver.

We designed a compact two-layer PCB with components mounted on both sides to fit within the limited space around the OBD2 ports found in modern vehicles. Bulk capacitors were strategically placed at each component's power pins to combat electrical noise and protect the vehicle's electronics. Additionally, we optimized trace lengths to further minimize interference.

Component Placeholders: During development, we included placeholders for potential future components, such as a standalone accelerometer, a Global Positioning System (GPS) module, and additional general-purpose input/output (GPIO) and power pins. We integrated these placeholders without increasing the PCB's size, providing flexibility for revisions or future additions. Initially, we planned to use separate standalone accelerometers and GPS modules, but ultimately, we decided to use the smartphone's onboard accelerometer and GPS capabilities, making these placeholders unnecessary for the final design.

Final PCB and Enclosure Design: The final PCB design measures 45mm square—compact enough to fit within most modern vehicles while housed in an enclosure. We created the design in Altium, and images of both sides of the board are shown in **Figure 4**.

We 3D-printed the enclosure using PLA, a renewable material, and designed it with Autodesk Fusion 360 to securely clamp around the OBD2 connector (Figure 1), holding the PCB in place. This sleek design ensures easy installation and compatibility with most modern vehicles, tailored to the constraints of the automotive environment.

Completed Scanner: Our hardware design combines efficiency, compatibility, and flexibility. From the ESP32-C3's BLE and TWAI capabilities to the reliable power regulation of the TPSM84205, each component plays a vital role in delivering reliable vehicle data to users. The compact PCB design and durable enclosure underscore our commitment to creating a user-friendly and practical solution for car enthusiasts and race teams. The finished scanner is shown in Figure 1.

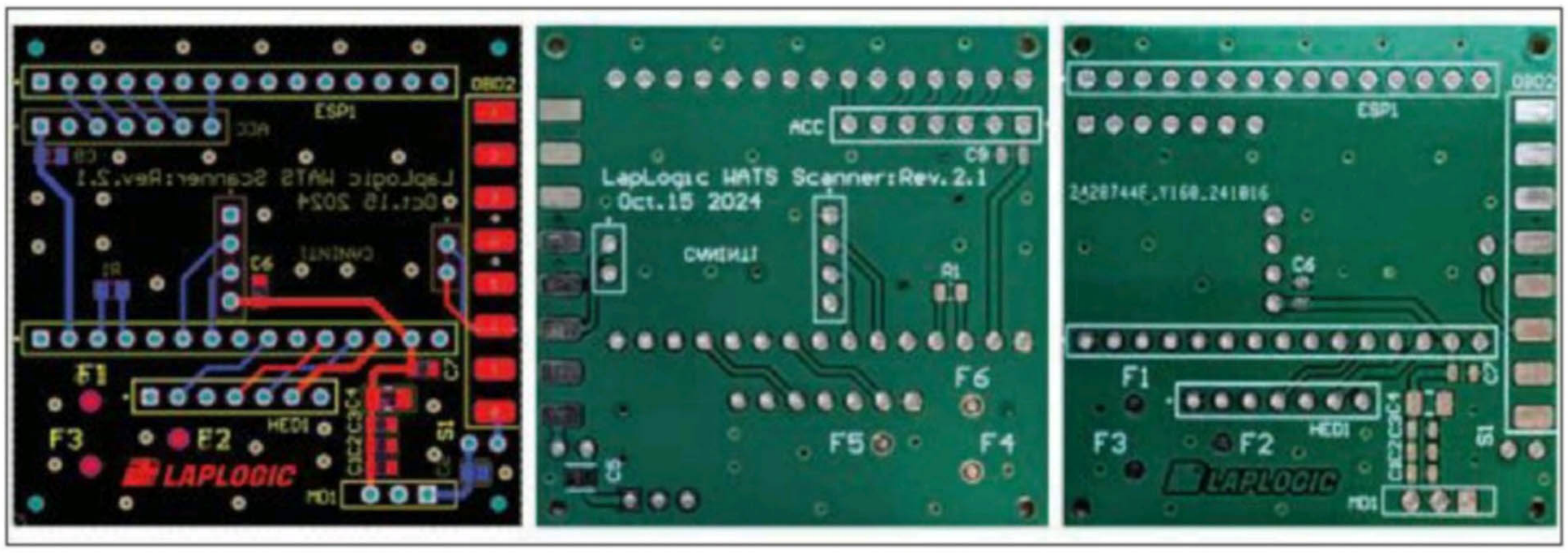


FIGURE 4
Three perspectives of the scanner's PCB. The left image displays the Altium preview, highlighting top-layer connections in red and bottom-layer connections in blue. The center image showcases the underside of the manufactured PCB. The right image features the top side, complete with our project logo.

CAN ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
7DF	02	01	0C	00	00	00	00	00
7E8	04	41	0C	1A	1B	00	00	00

TABLE 1
Example OBD2 engine RPM packet. The middle row displays the request sent by the microcontroller, and the bottom row shows the response from the vehicle received by the microcontroller.

FIRMWARE

When developing the firmware for the ESP32, our goal was to create efficient and adaptable code that would be easy to understand and maintain, while also being flexible for future changes. Below is an overview of the key stages of programming the ESP32.

Bluetooth & CAN Libraries: The process begins with the ESP32 broadcasting a Bluetooth signal, utilizing the ESP32 BLE Arduino library. This library manages all Bluetooth operations, including handling connections, advertising, and sending/receiving data. Once the ESP32 becomes discoverable, the mobile application can detect and establish a connection with the device. For CAN communication, we use the ESP32-TWAI-CAN library, which translates our requests into a format that the vehicle's system can understand.

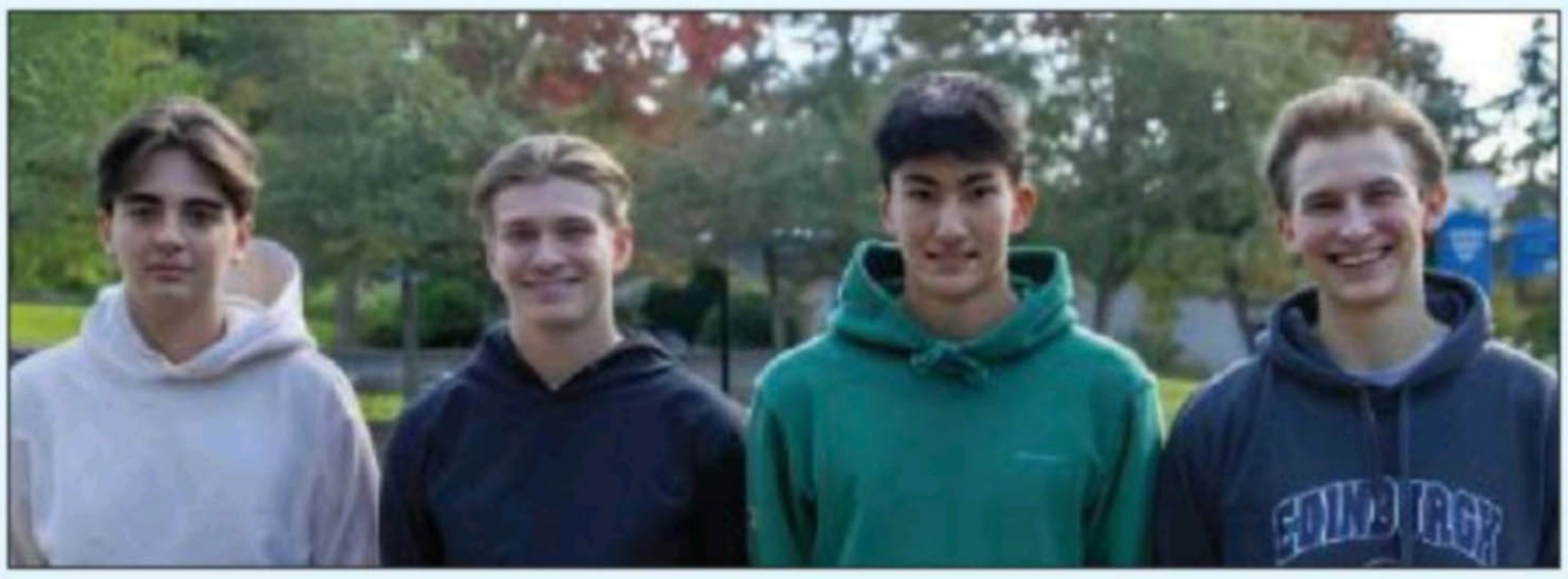
Requesting Supported PIDs: Once the mobile application establishes a Bluetooth connection with the ESP32, it sends a request for the vehicle's supported parameter IDs (PIDs). In response, the ESP32 queries the vehicle's CAN bus to retrieve the supported PIDs. Since there are up to 160 possible PIDs, and each vehicle supports a unique subset, it is essential to request only PIDs that the vehicle supports. After receiving the list from the vehicle, the ESP32 sends it to the mobile application, which displays the supported PIDs for the user. The user can then select a specific PID to view from the list.

Sending Data Requests to the Vehicle: When the user selects a PID to view, the app sends the corresponding PID code to the ESP32, instructing it to request the specific data from the vehicle. For instance, when querying PID 0C (engine speed), the ESP32 constructs a request packet tailored to the OBD2 protocol (see row 1 in **Table 1**). The packet begins with the CAN ID 7DF, which designates the message as a functional request, broadcast to all electronic control units (ECUs) on the vehicle's network. The next byte specifies the number of data bytes in the request—in this case, 02, since the packet includes 2 bytes of data beyond the header.

Following this, the packet includes the service mode 01, indicating a request for live data, and the PID code 0C, which identifies the specific parameter to query (engine speed). The remaining bytes are padded with zeros (00 00 00 00 00) to meet the OBD2 packet length requirements. Once prepared, the ESP32 waits for the OBD2 system to become ready, sends the packet, and listens for the vehicle's response.

Processing the Vehicle's Response: Row 2 in Table 1 shows what the response to our request might look like. The vehicle's response will always begin with 7E8, indicating that it is replying to the ESP32. The byte 04 indicates the length of the data in the response, which in this case is 4 bytes. The byte 41 is the response code indicating that live data is being returned from the vehicle. The 0C byte corresponds to the PID code, confirming that the data being returned pertains to engine speed (RPM). Finally, the bytes 1A 1B represent the engine speed data.

To process the data, the ESP32 combines the bytes 1A and 1B into a single number, resulting in the hexadecimal value 1A1B. This value is then converted into decimal, giving the



The LapLogic Team consists of four Electronics and Computer Engineering students from Camosun College. Pictured from left to right: Illia Pavych, Adam Schmidt, Alex Moore, and Matthew Pearson.

ABOUT THE AUTHORS

Illia Pavych — As Ilya approaches the final stage of his studies at Camosun College, he is ready to apply his expertise in software, hardware, PCB, and research to create a device that is accessible to all car enthusiasts. In his spare time, he enjoys car restoration, 3D printing, and skiing. Ipavych@gmail.com

Matthew Pearson — In addition to skills acquired at Camosun College, Matthew has developed extensive expertise in hardware and software through his role as a calibration technologist for the Department of National Defense. He aspires to create products for car enthusiasts that capture the same joy and fascination he experiences when learning about vehicles.

Alex Moore — During his time at Camosun, Alex has consistently kept his place on the Dean's Honour Roll. He excels in graphic design and mathematics and brings the experience of working on cars in his free time. Alex is an avid car enthusiast, but also enjoys sports, music, and the outdoors. alexkmoore16@gmail.com

Adam Schmidt — Adam first studied computer science at the University of Victoria for two years, gaining competency in Programming and Data Structures. He later continued his studies at Camosun College. In addition to his technical expertise, Adam brings a passion for motorsports and a drive to make the hobby more accessible to everyone. adam.s459@outlook.com

result of 6699. For engine speed, this number is divided by 4. So, 6699/4 = 1,674.75 RPM.

The remaining bytes 00 00 00 indicate that no additional data or parameters are being returned for this query. In some cases, these bytes may contain information about the availability of further data or parameters, but in this example, they are not relevant.

After the data has been processed, it is rounded to the nearest integer. The ESP32 then sends the result (1,675 RPM) over Bluetooth to the mobile app, where it is displayed for the user. Our application allows users to monitor up to 20 PIDs at one time.

SOFTWARE

App Overview: The purpose of our application is to act as the bridge between the user and the ESP32 device. By utilizing data retrieved by the ESP32 through OBD2 protocols and transmitted via BLE, the app processes and displays vehicle information. Additionally, it integrates GPS and accelerometer data to provide users with real-time insights into the vehicle's movement and location.

The app incorporates both mobile and remote functionalities, allowing users in the car and in the pits to access the same data. It uses MQTT as a messaging protocol to transmit and receive data between the CAR application, the server, and the REMOTE application, ensuring seamless communication without storing the information, itself. This approach delivers a comprehensive and intuitive experience tailored to the needs of car enthusiasts and race teams. Below, we outline the core functionality of the app and how it enhances the user experience.

.NET MAUI: This Multi-platform App UI enabled us to build cross-platform applications for Windows, Android, and

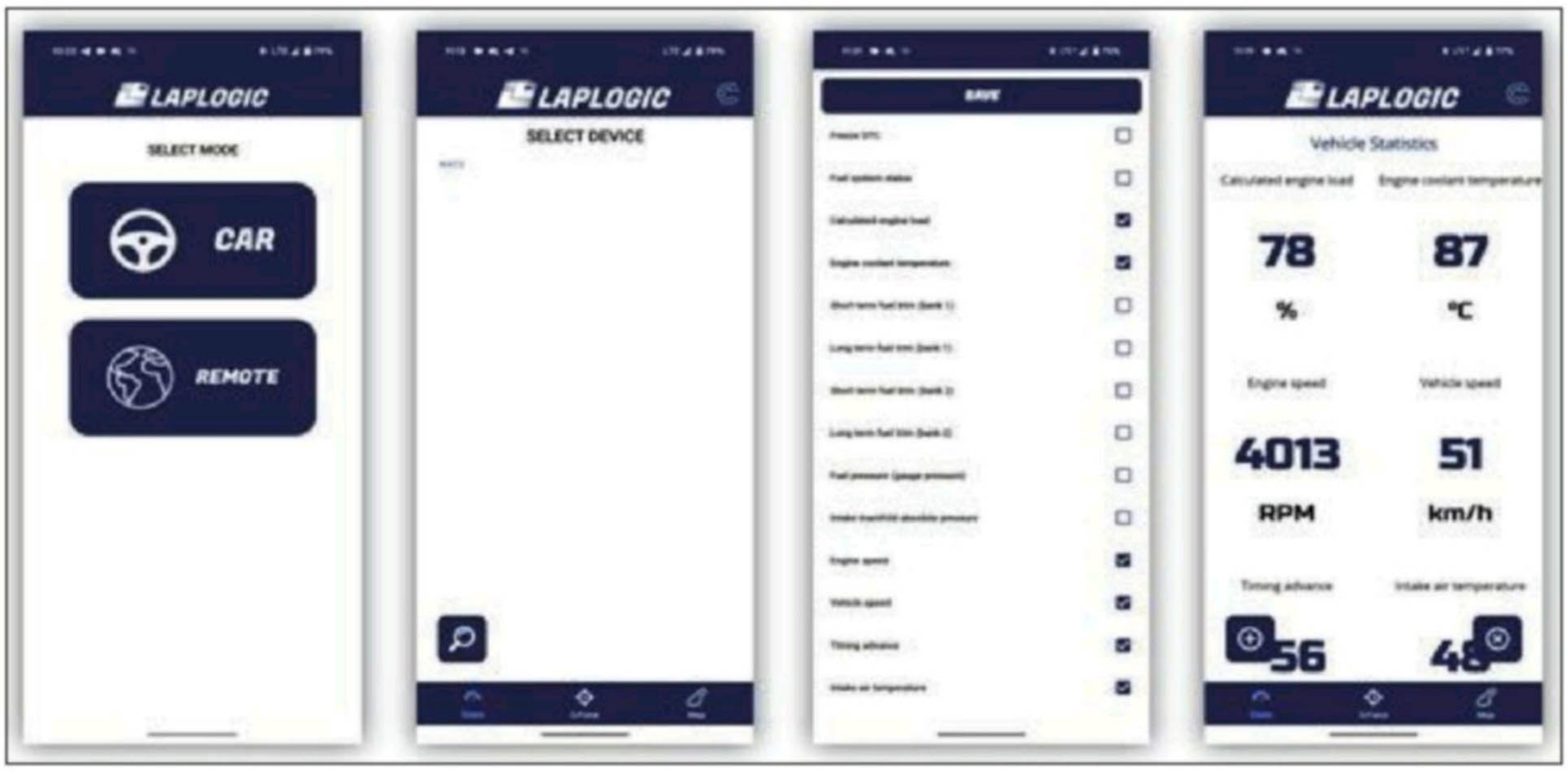
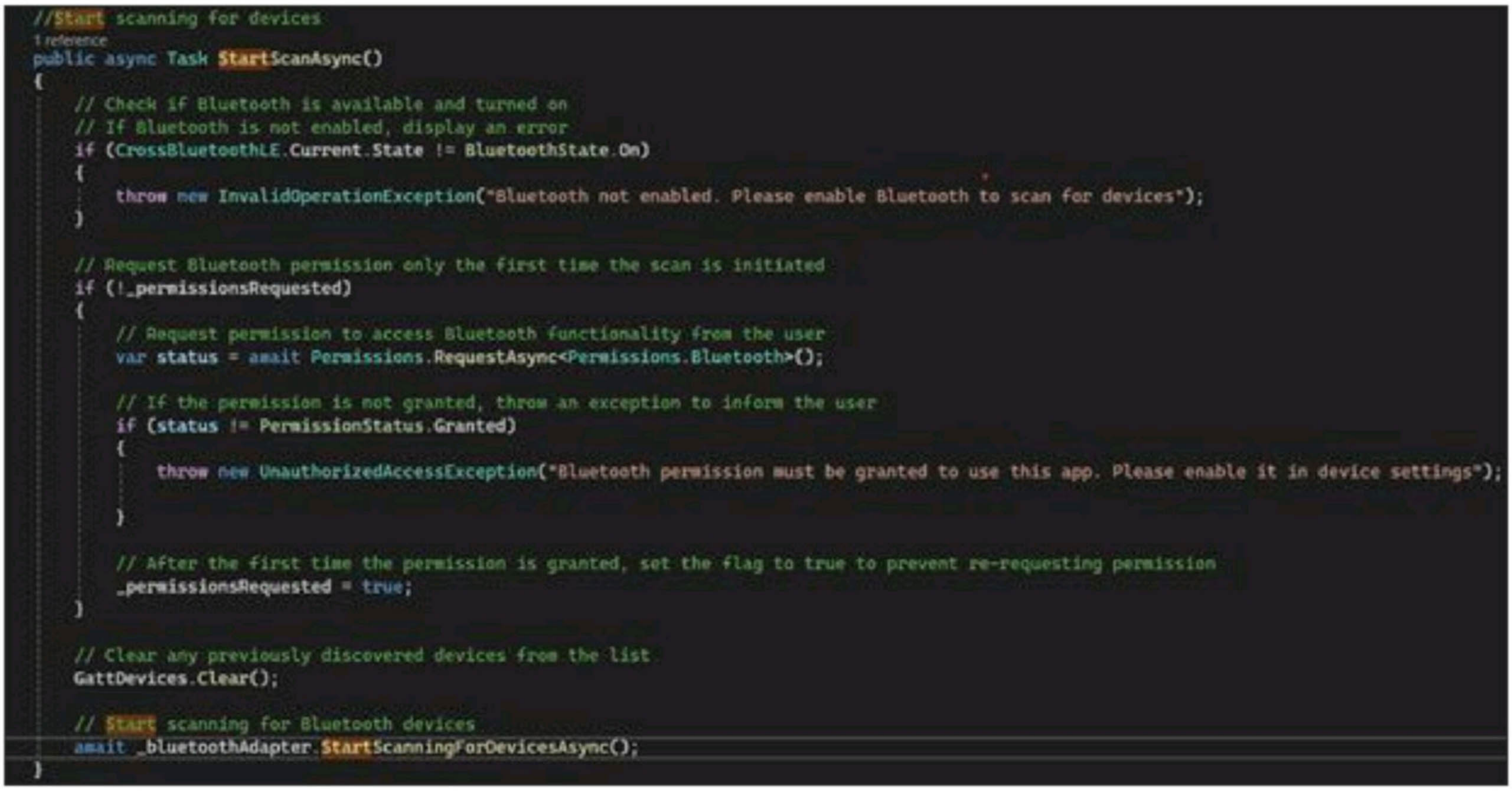


FIGURE 5
Multiple pages from the LapLogic application. From left to right, the app's homepage, where users select their desired mode; the Bluetooth device selection page, allowing users to scan and connect to our device; the statistics selection page, where users specify which statistics they want to view; the selected statistics in real time.



```
//Start scanning for devices
1 reference
public async Task StartScanAsync()
{
    // Check if Bluetooth is available and turned on
    // If Bluetooth is not enabled, display an error
    if (CrossBluetoothLE.Current.State != BluetoothState.On)
    {
        throw new InvalidOperationException("Bluetooth not enabled. Please enable Bluetooth to scan for devices");
    }

    // Request Bluetooth permission only the first time the scan is initiated
    if (!_permissionsRequested)
    {
        // Request permission to access Bluetooth functionality from the user
        var status = await Permissions.RequestAsync<Permissions.Bluetooth>();

        // If the permission is not granted, throw an exception to inform the user
        if (status != PermissionStatus.Granted)
        {
            throw new UnauthorizedAccessException("Bluetooth permission must be granted to use this app. Please enable it in device settings");

        }

        // After the first time the permission is granted, set the flag to true to prevent re-requesting permission
        _permissionsRequested = true;
    }

    // Clear any previously discovered devices from the list
    GattDevices.Clear();

    // Start scanning for Bluetooth devices
    await _bluetoothAdapter.StartScanningForDevicesAsync();
}
```

LISTING 1
This code ensures Bluetooth is enabled, requests necessary permissions, and begins scanning for devices. Once a connection has been established, the app can start exchanging data, such as vehicle performance, via Bluetooth.

iOS. This capability allowed us to design the app for Android devices, while maintaining the option to expand to iOS in the future. Additionally, it let us combine the CAR and REMOTE functions into a single app, simplifying development and ensuring a seamless user experience. We will now discuss the two core functions of the app, along with their specific features and uses.

CAR Mode: This portion of the application connects to the product via Bluetooth and streams real-time vehicle data, including vehicle performance and acceleration data based on the mobile device's orientation. It also displays the car's live location on a map. Data obtained by the mobile app is sent to the MQTT server for remote access.

REMOTE Mode: This portion of the application retrieves data from the MQTT server to monitor the vehicle's statistics, movement, and location in real time. It provides users with remote access to real-time updates, allowing them to track the vehicle from any location.

By using .NET MAUI, we combined these key functionalities into a single, cohesive application, making it easy for users to access both in-vehicle and remote data without needing to switch between multiple apps (**Figure 5**).

Bluetooth: Bluetooth is used in the CAR mode to connect the user's device to the product, allowing the app to request and receive real-time vehicle data. We use the Plugin.BLE library in .NET MAUI to handle Bluetooth functionality, which simplifies scanning for devices, establishing connections, and exchanging data through a straightforward API (Application Programming Interface). By leveraging this library, we can seamlessly interact with Bluetooth devices, ensuring efficient connections and data transfers.

For example, to scan for devices and request the necessary Bluetooth permissions, we use the code snippet shown in **Listing 1**, which demonstrates how the Plugin.BLE library is employed to handle these tasks.

GPS: The Global Positioning System (GPS) is the most used technology for determining precise locations anywhere on Earth. In our CAR mode, we use the GPS functionality of the mobile device, which is placed in the car, to display the vehicle's real-time location on a map. This location data is continuously updated and transmitted to our MQTT server, allowing the car's position to be tracked and displayed in the REMOTE mode. By integrating GPS data, the app allows users to monitor the vehicle's movement in real-time, providing valuable insights for both in-car and remote users. See **Figure 6** for an example of how the map is integrated in our application.

Accelerometer: An accelerometer detects the motion and orientation of a device. Our application leverages the onboard accelerometer in the user's mobile device to monitor the vehicle's movement in real-time. The accelerometer captures data to analyze the car's orientation and acceleration along the X, Y, and Z axes, which the app visualizes on a graph for detailed motion analysis.

The the app updates the graph in real-time and sends acceleration data to the MQTT server for remote monitoring, enabling continuous tracking of the vehicle's orientation and movement. Figure 6 shows the accelerometer interface.

MQTT: Message Queuing Telemetry Transport (MQTT) is a lightweight messaging protocol used for sending and receiving data between devices. In our project, we used MQTT to send vehicle data (telemetry, acceleration and location) between the device and the server. This data is organized into separate topics, allowing the remote application to retrieve and use it easily. An example of how we publish telemetry and location data to different MQTT topics is shown in **Listing 2**.

CONCLUSION

In just under three months, we developed a fully functional, wireless, automotive telemetry system. This included writing a program that communicates with a vehicle's ECU, creating an Android application capable of managing user input, connecting to a server, and displaying real-time data, as well as designing and assembling prototype devices featuring compact PCBs and custom 3D-printed enclosures

Although we accomplished much, there is always room for improvement. With more time, we would address minor bugs, refine the app's appearance for a polished, professional look, and implement additional features that didn't make it into the final version. What we achieved demonstrates significant potential, and we hope to continue refining the project beyond the classroom to create a product that truly meets the needs of car enthusiasts and race teams.

Entering the capstone semester, we carried a sense of dread fueled by stories from past students about its challenges and intensity. While some of those fears were realized—like spending a week trying to implement gauges, only to discover an unfixable library bug or facing setbacks with app crashes—there were also moments of triumph that made the experience worthwhile. Seeing our PCB work perfectly on the first try, and witnessing data successfully flow through our entire transmission system for the first time, were incredibly rewarding milestones.

Looking back, the capstone semester was a mix of stress, problem-solving, and personal growth. Despite its challenges, the experience has been incredibly rewarding. We're proud

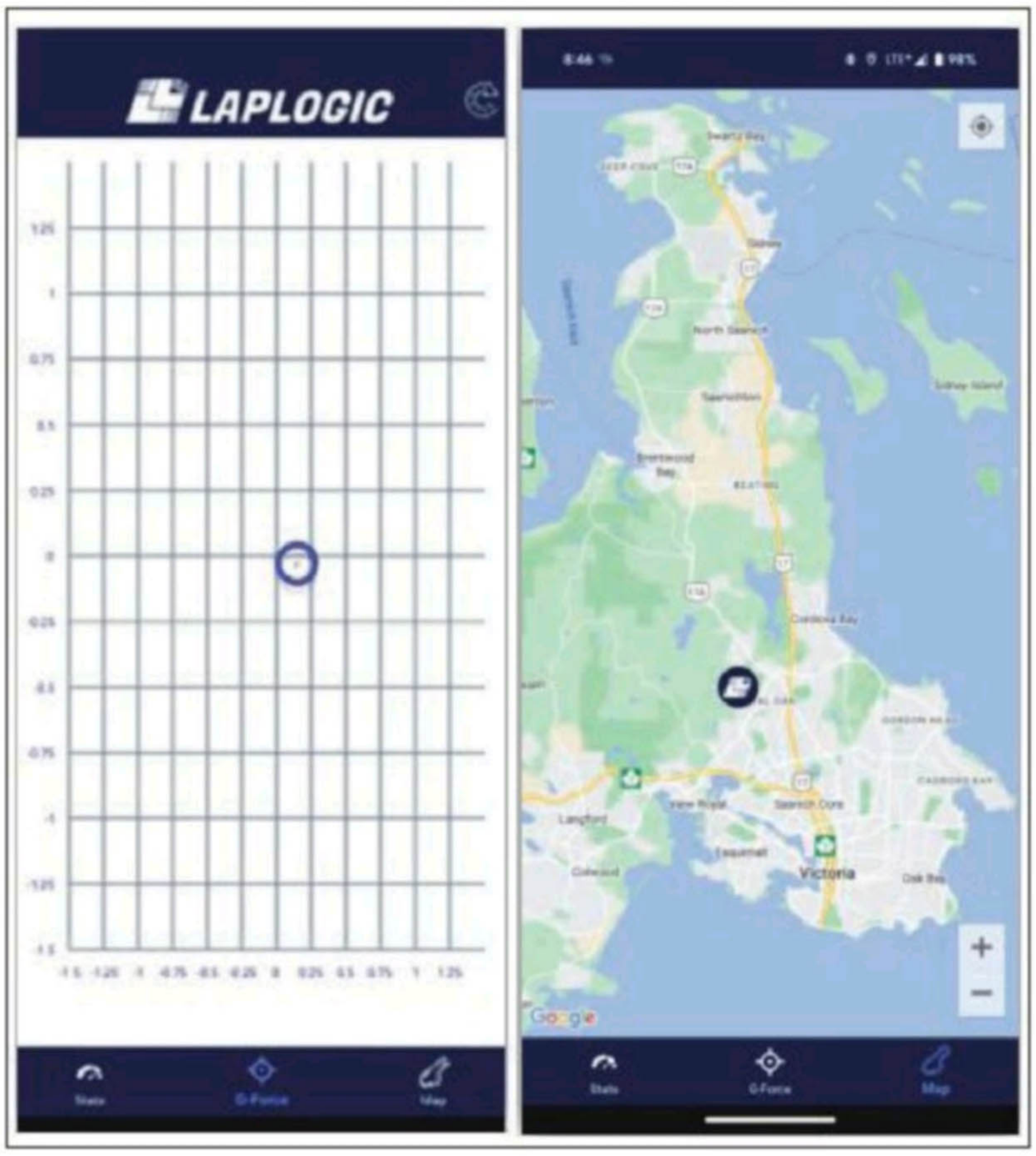
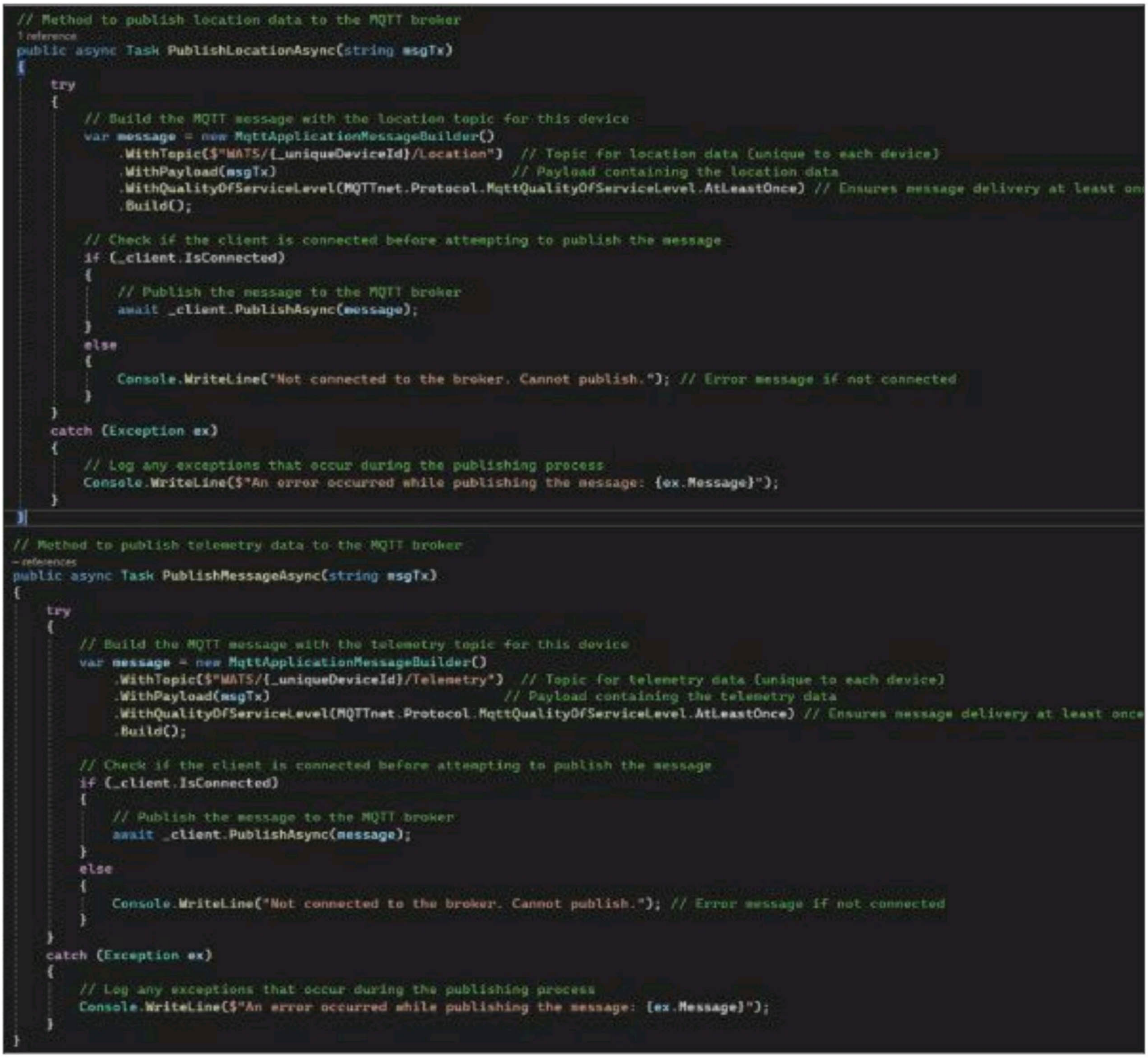


FIGURE 6
The Accelerometer and Map pages in the LapLogic application. The chart displays the car's motion. The map shows its real-time location.

of how far we've come, what we learned, and the practical system we built, which we believe can bring real value to the motorsport community.

Additional information about the LapLogic Wireless Automotive Telemetry System, including a User Manual, is available on the LapLogic website [1].



```
// Method to publish location data to the MQTT broker
1 reference
public async Task PublishLocationAsync(string msgTx)
{
    try
    {
        // Build the MQTT message with the location topic for this device
        var message = new MqttApplicationMessageBuilder()
            .WithTopic($"WATS/{_uniqueDeviceId}/Location")  // Topic for location data (unique to each device)
            .WithPayload(msgTx)                               // Payload containing the location data
            .WithQualityOfServiceLevel(MQTTnet.Protocol.MqttQualityOfServiceLevel.AtLeastOnce) // Ensures message delivery at least on
            .Build();

        // Check if the client is connected before attempting to publish the message
        if (_client.IsConnected)
        {
            // Publish the message to the MQTT broker
            await _client.PublishAsync(message);
        }
        else
        {
            Console.WriteLine("Not connected to the broker. Cannot publish."); // Error message if not connected
        }
    }
    catch (Exception ex)
    {
        // Log any exceptions that occur during the publishing process
        Console.WriteLine($"An error occurred while publishing the message: {ex.Message}");
    }
}

// Method to publish telemetry data to the MQTT broker
- references
public async Task PublishMessageAsync(string msgTx)
{
    try
    {
        // Build the MQTT message with the telemetry topic for this device
        var message = new MqttApplicationMessageBuilder()
            .WithTopic($"WATS/{_uniqueDeviceId}/Telemetry")  // Topic for telemetry data (unique to each device)
            .WithPayload(msgTx)                                // Payload containing the telemetry data
            .WithQualityOfServiceLevel(MQTTnet.Protocol.MqttQualityOfServiceLevel.AtLeastOnce) // Ensures message delivery at least once
            .Build();

        // Check if the client is connected before attempting to publish the message
        if (_client.IsConnected)
        {
            // Publish the message to the MQTT broker
            await _client.PublishAsync(message);
        }
        else
        {
            Console.WriteLine("Not connected to the broker. Cannot publish."); // Error message if not connected
        }
    }
    catch (Exception ex)
    {
        // Log any exceptions that occur during the publishing process
        Console.WriteLine($"An error occurred while publishing the message: {ex.Message}");
    }
}
```

LISTING 2
By publishing the data to separate topics, we ensure that vehicle data (location, telemetry, and acceleration) is organized such that the remote application can easily retrieve and use the data, ensuring efficient communication and real-time monitoring.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials

References [1] as marked in the article can be found there.

RESOURCES

Altium | https://www.altium.com
Arduino | https://www.arduino.cc
Autodesk | https://www.autodesk.com/ca-en
CSS Electronics | https://www.csselectronics.com
EMQX | https://www.emqx.com/en
Espressif Systems | https://www.espressif.com
JLCPCB | https://jlcpcb.com
MQTT | https://mqtt.org
Texas Instruments | https://www.ti.com
Ultimaker | https://ultimaker.com

FREE Download Now!
Mac, Windows and Linux

Blackmagicdesign

DaVinci Resolve
File
Edit
Trim
Timeline
Clip
Mark
View
Playback
Fusion
Color
Fairlight
Workspace
Help
Media Pool
Sync Bin
Transitions
Titles
Effects
Aerospace Propulsion Project
Quick Export
Full Screen
Inspector
Master / Clips
Main Edit
01:00:12:17
DaVinci Resolve 19
Media
Cut
Edit
Fusion
Color
Fairlight
Deliver

Edit and color correct using the same software used by Hollywood, for free!

DaVinci Resolve is Hollywood's most popular software! Now it's easy to create feature film quality videos by using professional color correction, editing, audio and visual effects. Because DaVinci Resolve is free, you're not locked into a cloud license so you won't lose your work if you stop paying a monthly fee. There's no monthly fee, no embedded ads and no user tracking.

Editing, Color, Audio and Effects!
DaVinci Resolve is the world's only solution that combines editing, color correction, visual effects, motion graphics and audio post production all in one software tool! You can work faster because you don't have to learn multiple apps or switch software for different tasks. For example, just click the color page for color, or the edit page for editing! It's so incredibly fast!

Professional Editing
DaVinci Resolve is perfect for editing sales or training videos! The familiar track layout makes it easy to learn, while being powerful enough for professional editors. You also get a library full of hundreds of titles, transitions and effects that you can add and animate! Plus, DaVinci Resolve is used on high end work, so you are learning advanced skills used in TV and film.

Creative Color Correction
DaVinci Resolve's color page is Hollywood's most advanced color corrector and has been used on more feature films and television shows than any other system! It has exciting new features to make it easier to get amazing results, even while learning the more advanced color correction tools. There's PowerWindows™, qualifiers, tracking, advanced HDR grading tools and more!

Designed to Grow With You
DaVinci Resolve is designed for collaboration so as you work on larger jobs you can add users and all work on the same projects, at the same time. You can also expand DaVinci Resolve by adding a range of color control panels that let you create unique looks that are impossible with a mouse and keyboard. There's also edit keyboards and Fairlight audio consoles for sound studios!

DaVinci Resolve 19	**Free**
DaVinci Resolve Micro Color Panel	**Only $509**

Learn the basics for free then get more creative control with our accessories!

Learn more at www.blackmagicdesign.com

Download free on the
DaVinci Resolve website

NO SUBSCRIPTIONS • NO ADS • NO USER TRACKING • NO AI TRAINING

Wake-Up Receivers

Extending Operational Periods in Battery-Powered Applications

By
Abel Didouh

Wake-up receivers allow battery-powered IoT devices to conserve battery life by remaining in a standby state until needed. In this article, Abel discusses the importance of wake-up receivers for achieving low energy consumption, and describes how he built a demonstration prototype with an integrated RFicient wake-up receiver on the PCB.

Power consumption is a crucial criterion for the Internet of Things (IoT). Ongoing wireless connectivity uses a considerable amount of battery power, which usually restricts the operational lifespan of wireless sensor nodes. The wake-up receiver creates new opportunities by allowing continuous access to data and real-time information with low latency and minimal power usage.

In this article I write about wake-up receivers. I provide an overview of the current state of the art in wake-up receivers, and also explore several use cases that have the potential to be developed into fully integrated solutions. I then show you how to integrate and develop a solution around a wake-up receiver.

OK, let's explore the world of wake-up receivers!

ADDRESSING KEY IOT CHALLENGES WITH A WAKE-UP RECEIVER

Continuous monitoring of a wireless channel is critical for applications where the exact timing of an incoming radio message is unknown to the receiver. Once a radio request has been received, a rapid response is expected. The wake-up receiver plays a key role in unlocking the full potential of the IoT.

Many IoT applications demand a delicate balance between ultra-low power consumption and the ability to "wake up" the device upon receiving a radio message. Traditional solutions typically rely on periodically waking up the receiver to detect specific messages, which still leads to significant power consumption on both the transmitter and receiver sides. This ongoing power drain can limit the scalability and effectiveness of IoT systems, especially in scenarios where devices are expected to remain in low-power states for extended periods.

The wake-up receiver offers a more efficient method, by enabling devices to stay in ultra-low power modes while still being able to "wake up" quickly when necessary, thereby significantly reducing energy usage. This technology ensures that the IoT device can remain in a dormant state until a relevant radio signal is detected, minimizing unnecessary power consumption. This result improves battery life and extended operational periods, which is particularly important in remote or battery-powered applications.

By incorporating wake-up receivers, IoT systems can not only achieve low energy consumption but also maintain high responsiveness, ultimately unlocking the full potential of IoT solutions. The combination of low-power operation and instant wake-up capabilities makes this technology indispensable for a wide range of IoT applications.

Can you imagine having to change the battery of a connected device every year? I definitely can't!

As you can probably guess, the wake-up receiver can be used in a wide range of practical applications, for example, smart homes (security and monitoring systems), logistics (asset tracking and inventory management), automotive (keyless entry and diagnostics), industry (wireless networks), retail (smart shelves and labels), and healthcare (fitness monitoring, intelligent implants). It helps create smarter and more efficient systems across all these sectors.

Don't worry, I'm writing for a magazine that specializes in embedded systems and electronics. In the next section, we'll dive into the heart of the technology.

WHAT IS A WAKE UP RECEIVER?

A wake-up receiver is a specialized device designed for low power consumption and low data rates, typically reaching up to a few kilobits per second. Its primary function is to activate the connected host system.

Several wake-up receivers are available on the market. Of course, each one has a specific reception frequency for the listening channel. Here are two wake up receivers that are commonly used.

RFicient: The first wake-up receiver I'll be using was developed by the Fraunhofer Institute for Integrated Circuits IIS in Germany. It's called RFicient and is distributed by EBV Elektronik [1]. This component is designed to listen for wake-up signals on the following frequency bands: 433MHz, 868MHz, and 2.4GHz. For those familiar with the technology, it operates within the license-free frequency bands. The supply current of this receiver is around 3μA.

STM32WL3x: Another wake-up receiver that is integrated directly into the microcontroller and forms a system-on-chip (SoC) is the STM32WL3x by STMicroelectronics. This SoC includes a wake-up receiver as a secondary radio with a low-power, autonomous wake-up receiver, consuming as little as 4μA in always-on mode. For enhanced flexibility, it supports a wide range of frequency bands from 100MHz to 2.4GHz.

In this article, I will focus on the integration of the RFicient wake-up receiver.

RFICIENT: THE HEART OF THE COMPONENT

The RFicient has a superheterodyne structure. If we unpack the wake-up receiver, we might find a low-noise amplifier, an oscillator, a mixer, an intermediate-frequency filter, an intermediate frequency amplifier and an envelope-detection ASK demodulator (**Figure 1**).

Why use a superheterodyne structure? This structure is commonly used in radio receivers and other communication devices for several reasons. This principle, invented by Edwin Armstrong in 1918, improves the performance of receivers by simplifying design and increasing sensitivity. The operation of a superheterodyne receiver relies on converting the incoming signal (usually high frequency, or HF) into a lower and more stable intermediate frequency (IF). This is achieved through a local oscillator that generates a specific frequency, and a mixer that combines this frequency with the input signal's frequency. The resulting IF is easier to process, because it typically falls within a range that optimizes the performance or filters and amplifiers.

The use of an IF offers several advantages. First, it allows for efficient separation of the useful signal from noise and interference, improving signal quality. Next, the IF is easier to amplify and filter using readily available electronic components. Furthermore, this system provides better control of the tuning frequency, since the receiver can operate independently of the carrier frequency of the input signal.

WHERE DO I PLACE THIS COMPONENT ON THE BOARD?

The wake-up receiver can be strategically positioned between the antenna and the microcontroller, serving as a key

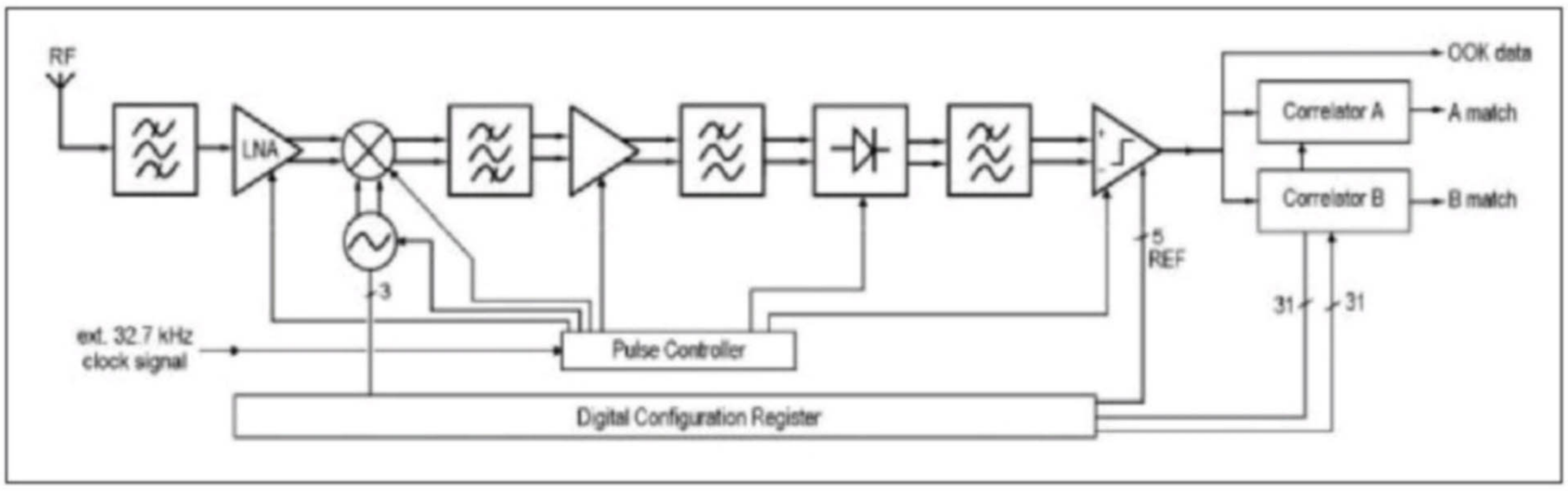


FIGURE 1
Superheterodyne architecture.
(Source: RFicient Datasheet [2])

intermediary (**Figure 2**). Its main functions are to monitor incoming signals and activate the microcontroller when it detects a wake-up frame.

During my internship at ALCIOM, a design office specializing in electronic development, I designed a PCB that will be used to demonstrate the capabilities of RFicient products and integrated the wake-up receiver on a PCB. The board's architecture is shown in the block diagram in **Figure 3**. (Editor's Note: The ALCIOM name is probably familiar to many readers, because its founder, Robert Lacoste, wrote "The Darker Side," a column in *Circuit Cellar*, from 2007-2021.)

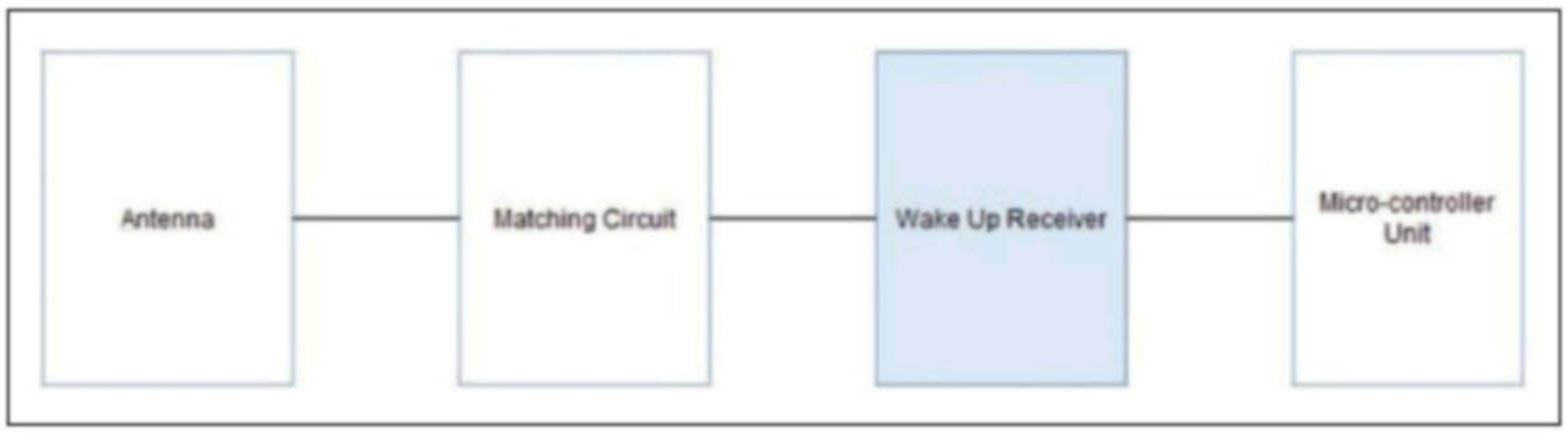

Antenna
Matching Circuit
Wake Up Receiver
Micro-controller Unit

FIGURE 2
Placement of a wake-up receiver in a radio frequency system. (Source: ALCIOM [3])

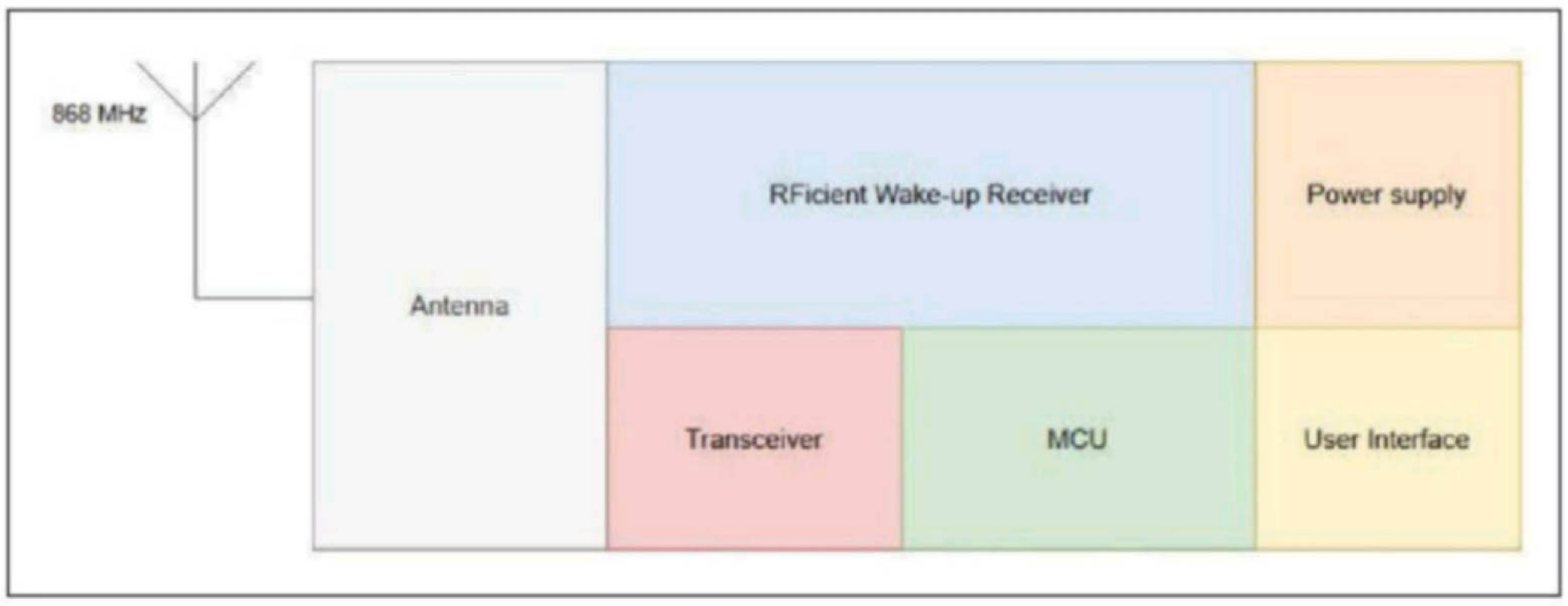

868 MHz
Antenna
RFicient Wake-up Receiver
Power supply
Transceiver
MCU
User Interface

FIGURE 3
Block diagram of the author's demonstration prototype with an integrated wake-up receiver on its PCB. (Source: ALCIOM [3])

The development board is structured into several key sections to ensure optimal functionality. The power supply section provides stable voltage levels to all components. The radio section handles signal reception and transmission. The RFicient module is dedicated to wake-up signal processing, enhancing low-power performance. Finally the user interface section facilities interaction and control, making the system accessible and easy to use. I implement UART communication between the board and the computer. The transceiver enables both data transmission and reception.

The complete architecture is shown in **Figure 4**.

ALCIOM chose to work with the 868MHz frequency band, which is a public band. The RFicient communicates with the MCU using the SPI protocol. For the transceiver section, I selected the Texas Instruments CC1201.

After making these decisions, I eagerly turned to my CAD software. At the office, I use Proteus for schematic design. The schematic design of the development board involved several key steps to ensure functionality and minimize issues.

First, I considered the typical schematics for each component to ensure proper integration. Second, using Proteus, I created custom footprints for the components to ensure they were accurately represented in the design. Third, after the components were defined, I focused on their placement,

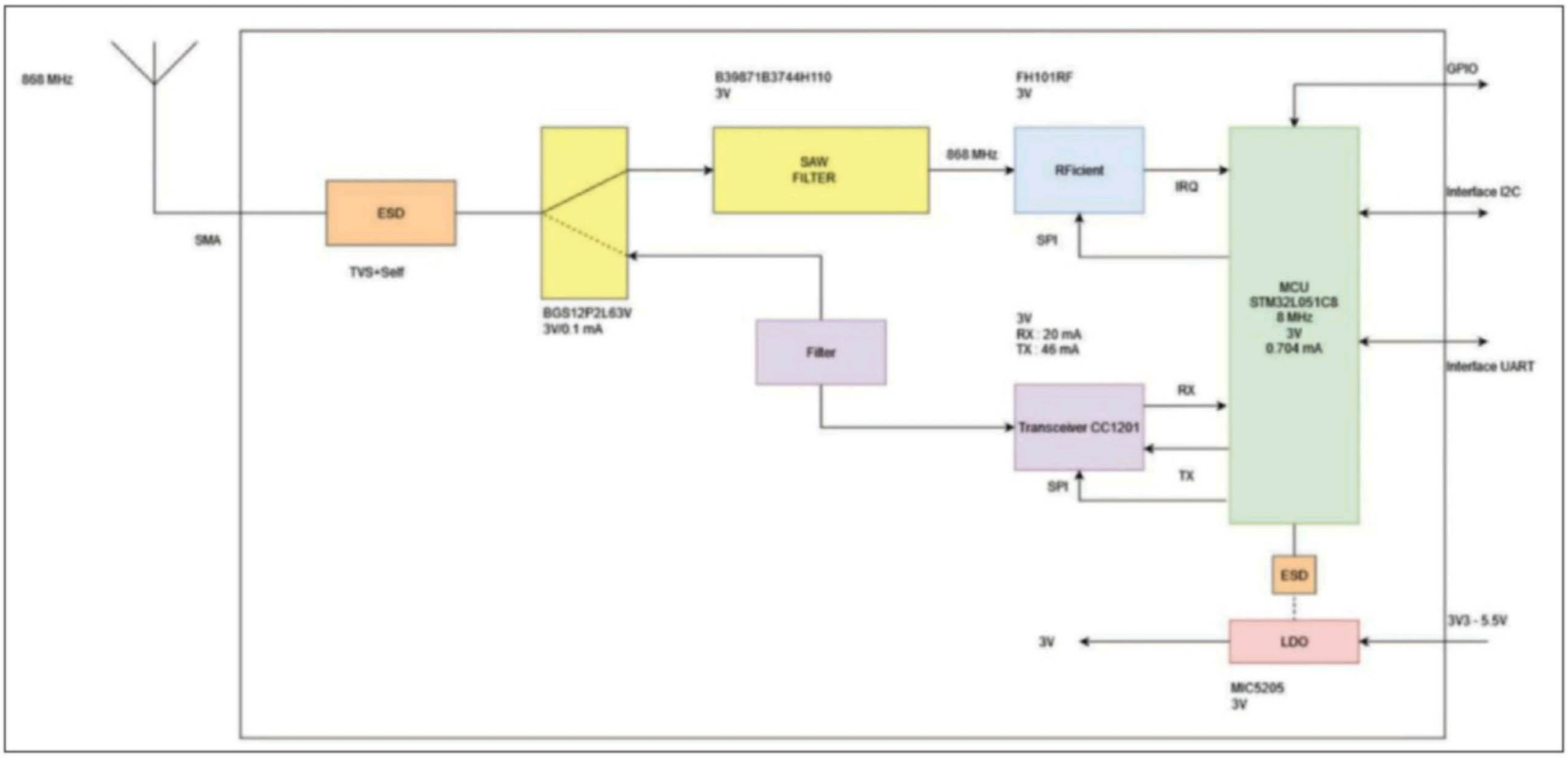

868 MHz
B39871B3744H110
3V
FH101RF
3V
GPIO
SAW
FILTER
868 MHz
RFicient
IRQ
Interface I2C
ESD
SMA
TVS+Self
SPI
BGS12P2L63V
3V/0.1 mA
Filter
3V
RX : 20 mA
TX : 46 mA
MCU
STM32L051C8
8 MHz
3V
0.704 mA
Interface UART
RX
Transceiver CC1201
TX
SPI
ESD
3V3 - 5.5V
3V
LDO
MIC5205
3V

FIGURE 4
Complete architecture of the author's demonstration prototype with an integrated wake-up receiver on its PCB. (Source: ALCIOM [3])

carefully grouping them to minimize parasitic effects and to reduce interference between different sections of the circuit. This step is critical in maintaining signal integrity and optimizing the performance of the board. Fourth, when it came to routing the PCB, I followed a structured approach to ensure the optimal path for the traces. Horizontal traces were routed on the "Top" layer, while vertical traces were placed on the "Bottom" layer. This separation helps reduce cross-talk and keeps the design clean, ultimately improving

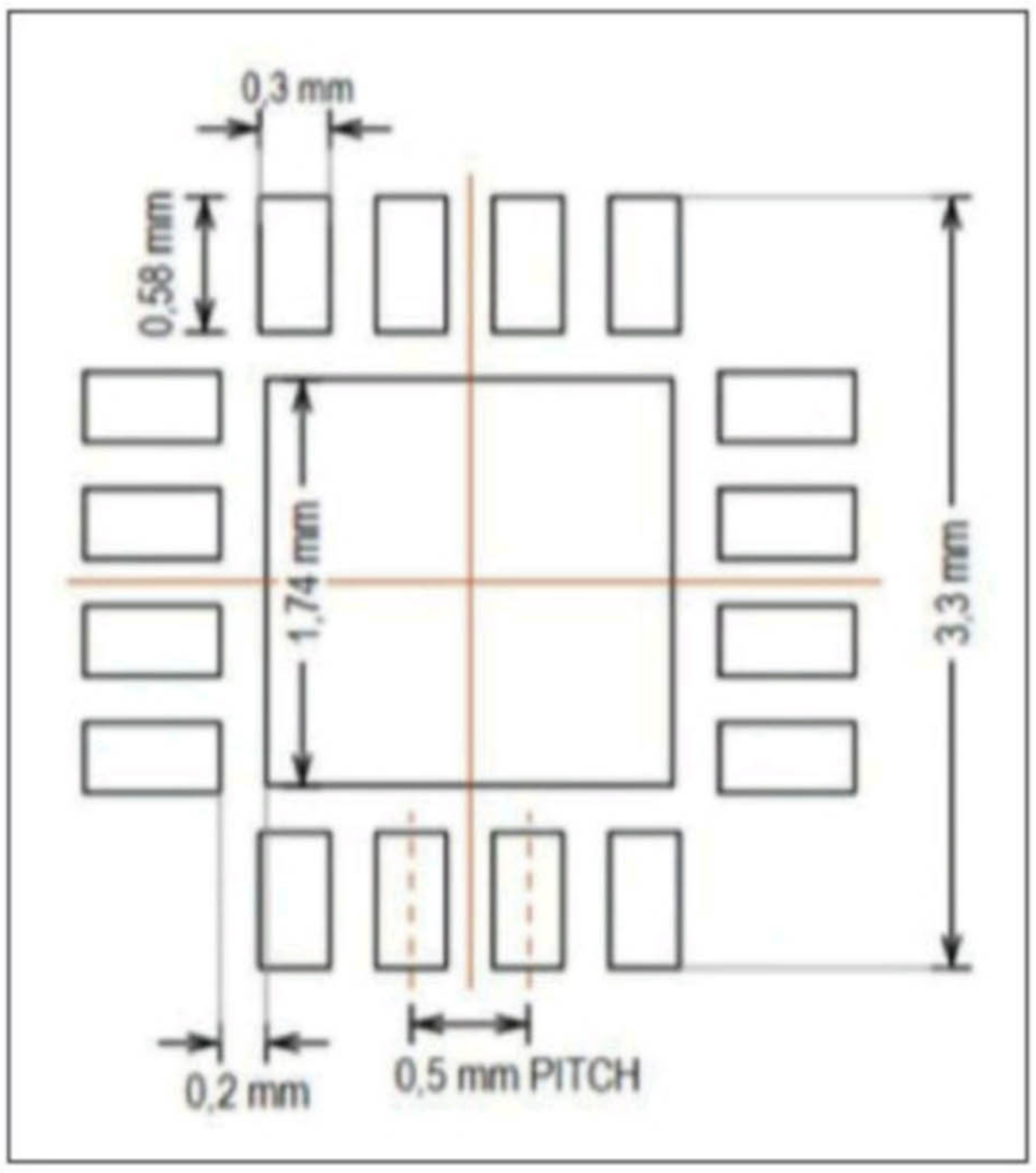

0,3 mm
0,58 mm
1,74 mm
3,3 mm
0,2 mm
0,5 mm PITCH

FIGURE 5
RFicient footprint. (Source: RFicient Datasheet [2])

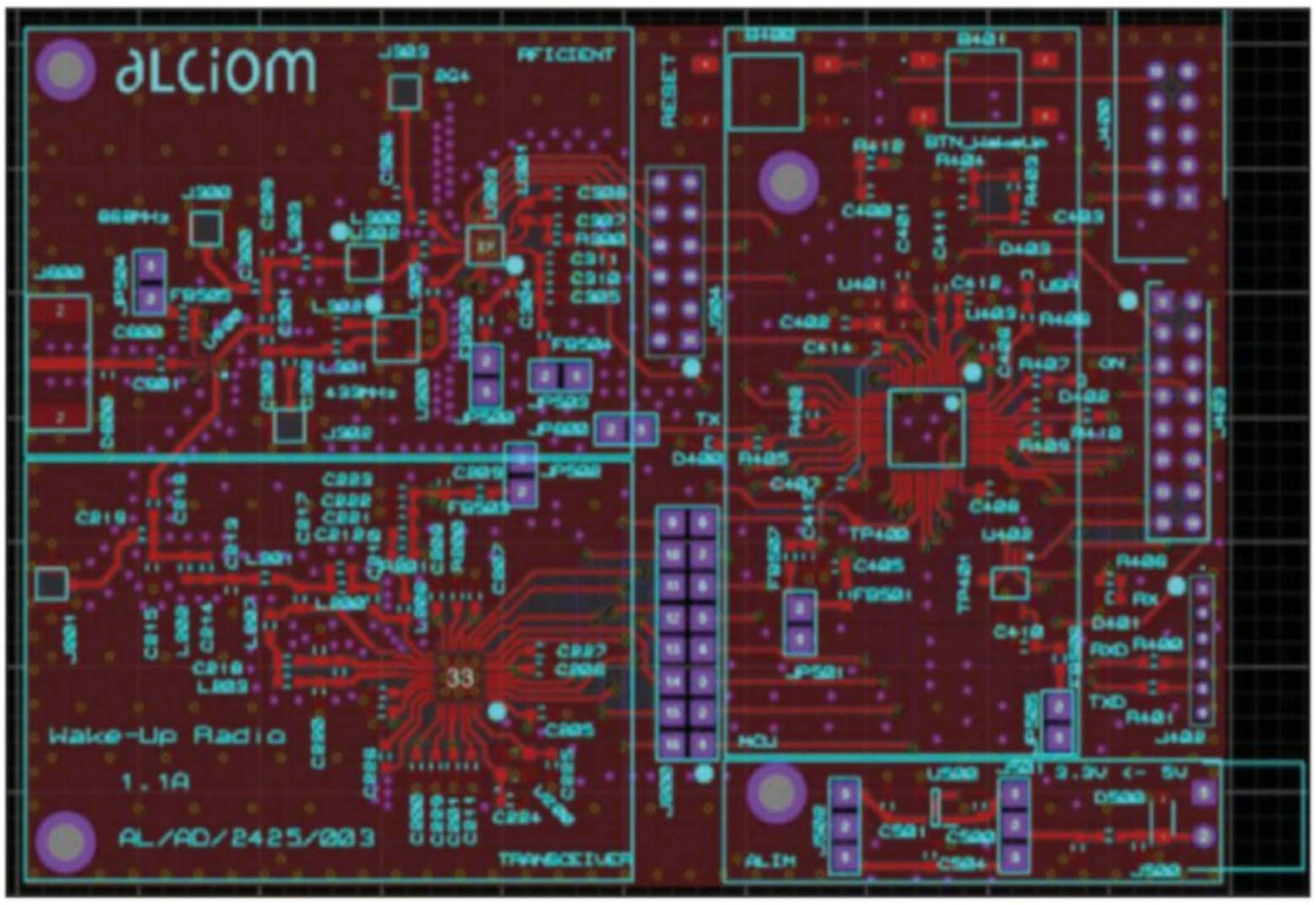
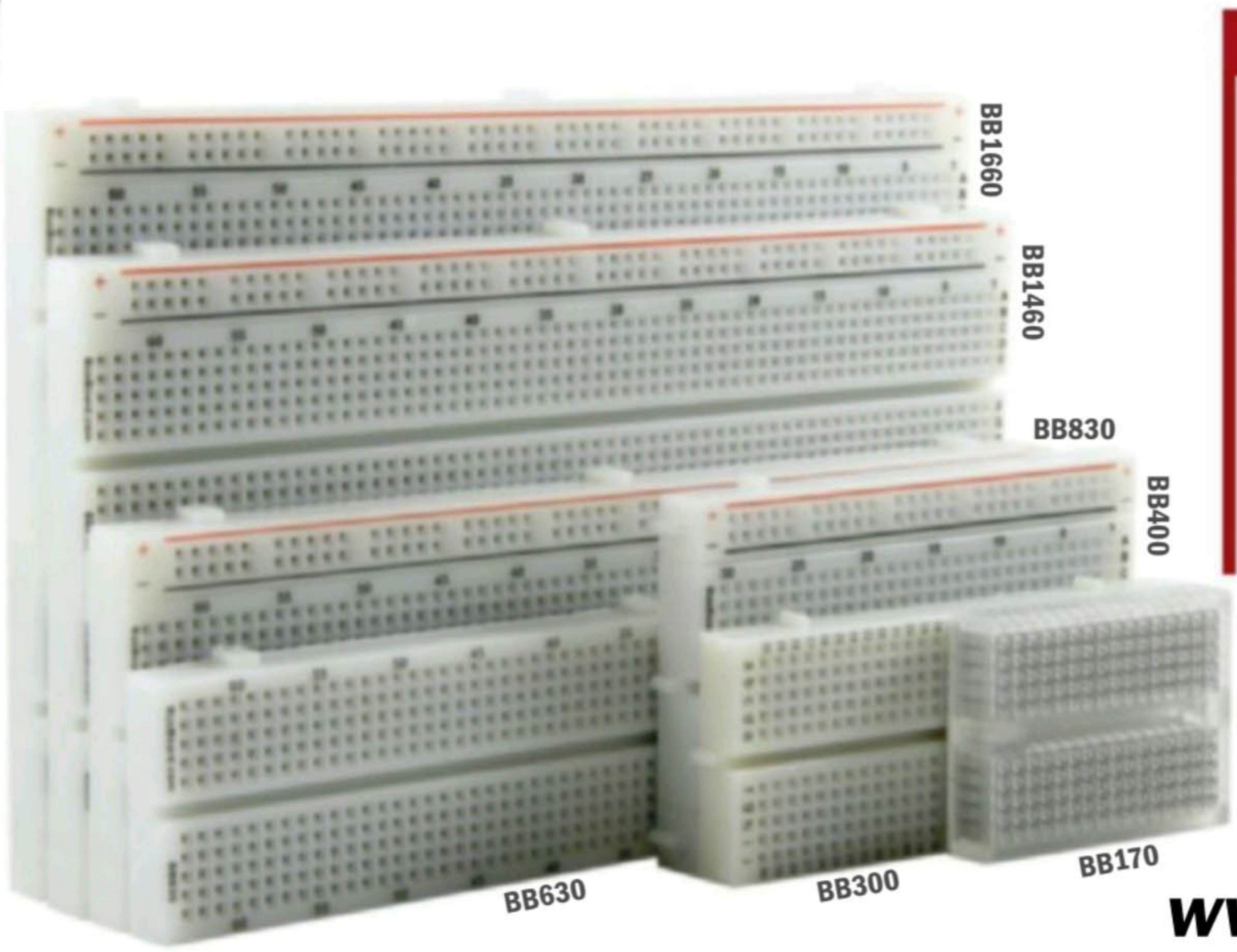

aLCiOM
RFICIENT
RESET
Wake-Up Radio
1.1A
AL/AD/2425/003
TRANSCEIVER
ALIM

FIGURE 6
The "Top" layer of the author's PCB. (Source: ALCIOM [3])

Experience the
BREADBOARD DIFFERENCE

BPS™ BusBoard Prototype Systems

Enjoy prototyping with high reliability precision breadboard contacts and avoid the frustration of poor breadboard connections. Usable with square-post headers and a range of wire sizes down to 26AWG.


BB1660
BB1460
BB830
BB400
BB630
BB300
BB170

Available for purchase at your trusted retailer

MOUSER ELECTRONICS
DigiKey
ARROW
JAMECO ELECTRONICS
amazon.com

For more details on BusBoard breadboards and prototyping PCBs visit:
www.BusBoard.com/CC

the overall functionality and reliability of the development board.

The footprint of the RFicient is shown in **Figure 5**, and the "Top" layer is shown in **Figure 6.**

To validate the footprints of the critical components in the board design, I followed a meticulous process (**Figure 7**). First, I printed the PCB layout to visually check the arrangement of the components on the board. This allowed me to identify any potential errors before final assembly. Then, to ensure the accuracy of the footprints, I used a microscope to carefully inspect every detail.

Next, I proceeded to solder the components. After many hours of soldering, I ended up with the boards shown in **Figure 8**. The process took time and precision, but it was rewarding to see the components properly connected. Each of the steps—from carefully placing the components to ensuring the solder joints—were solid, required great patience. Once everything was in place and the soldering was complete, I inspected the boards to make sure they were functional and well-built. It was a long but satisfying journey, but at its conclusion, I had the finished boards ready for the next steps in the project.

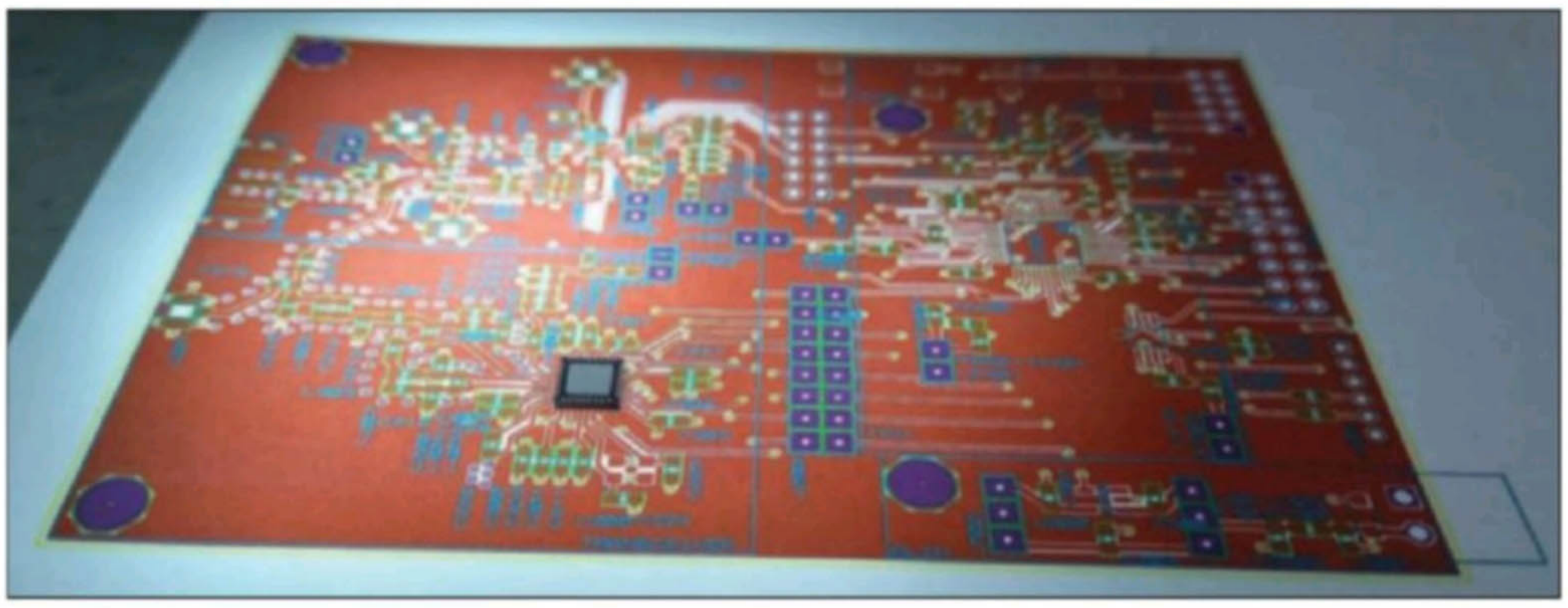


FIGURE 7
Footprint validation. (Source: ALCIOM [3])

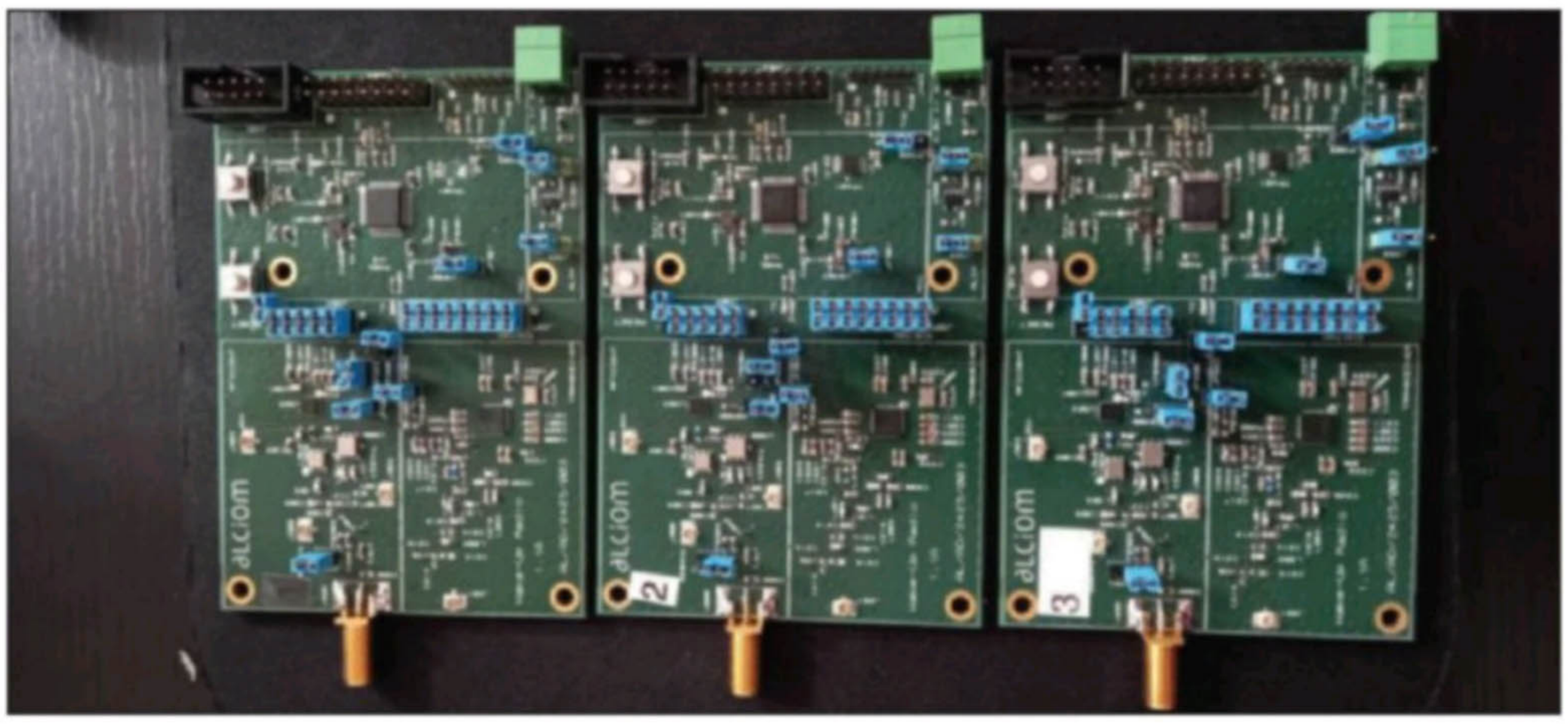


FIGURE 8
Complete board integrating the wake-up receiver. (Source: ALCIOM [3])

Additional materials from the author are available at:
www.circuitcellar.com/article-materials

References [1] to [3] as marked in the article can be found there.

RESOURCES

ALCIOM | www.alciom.com

HOW CAN I WAKE UP MY DEVICE?

When the connected device is in sleep mode, it remains inactive until it receives a wake-up signal. To activate the device, a wake-up frame must be transmitted. This frame contains a wake-up word, which is encoded using a 31-bit pattern, ensuring precise identification. The wake-up word is critical for distinguishing valid wake-up signals from other potential interference or noise in the environment.

To transmit this signal, the On-Off Keying (OOK) modulation technique is employed. This modulation method is simple, yet effective, allowing the wake-up frame to be received and processed efficiently by the device. The demodulation in OOK involves detecting the presence or absence of a signal to decode the transmitted data. In this specific case, correlation pattern A corresponds to the value 0 (0x6ACD674F), while correlation pattern B represents the value 1 (0x6D389773). These values help determine the state of each bit in the received data frame.

The frame identifier (ID) is 0x7DA8, which translates to the binary sequence 0b0111 1101 1010 1000. This sequence is then broken down into a bit pattern: ABBB BBAB BABA BAAA. This binary pattern can be used to identify specific motifs in the transmitted data, with each "A" and "B" representing distinct states related to the OOK modulation. The complete frame consists of 68 bytes of data.

I used this architecture composed of two boards. The first board plays the role of the transmitter, while the second board is the receiver board.

As you can see in **Figure 9**, which shows the structure of the payload, we have a time constraint. The preamble must be sent with a low data rate, whereas the ID block must be sent with a high data rate. Therefore, our transmitter must have two radio configuration profiles.

THE JOY OF EMBEDDED DEVELOPMENT

During my internship, I had the opportunity to develop embedded software for the wake-up receiver. My work was based on the technical documentation of the FH101RF RFicient component. The project required a structural approach. The first step involved creating two essential files: FH101RF.c and FH101RF.h. These files served as a backbone of the software, integrating all the necessary registers of the FH101RF component. The .h

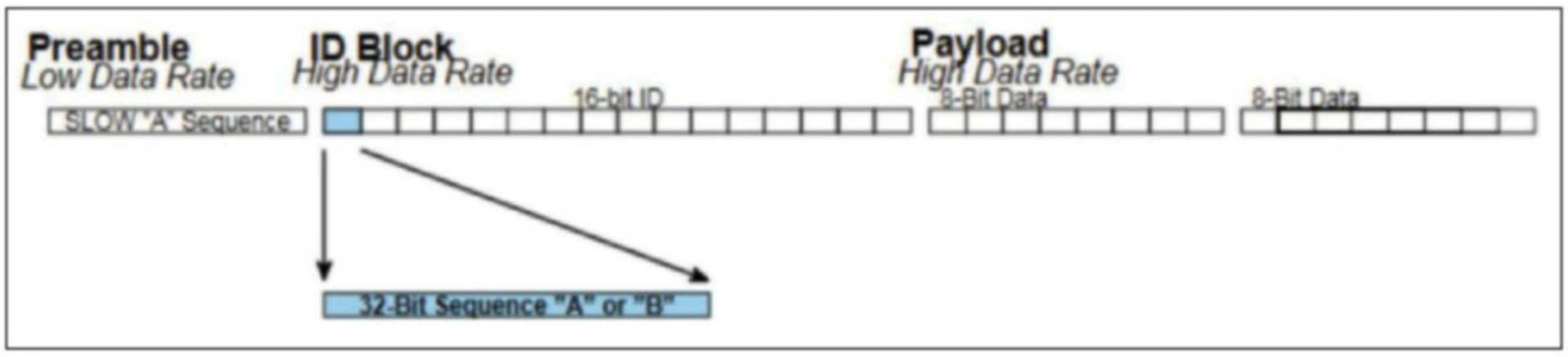


FIGURE 9
Structure of a payload.

file primarily contains the register definitions, constants, and function prototypes, while the .c file includes the implementation of these functions. By structuring the code this way, I ensured better modularity and maintainability, making it easier to debug and expand if needed.

Once the register integration was complete, I focused on developing the power-up routine in C. The first procedure calibrates the component's internal oscillator to achieve the appropriate radio frequency. The second procedure, know as "Sample Pulse Length," adjusts the internal time references to ensure precise signal timing. The third, the "Comparator Offset" procedure, compensates for internal voltages offsets, which is essential for maintaining RF sensitivity.

I also developed the embedded software for the transmitter. To configure each register based on the radio parameters, I used SmartRF, a software package from Texas Instruments. The configuration was tailored to the preamble section and the ID section, ensuring optimal performance and compatibility. By leveraging SmartRF, I was able to fine-tune the transmission settings efficiently, adjusting key parameters such as frequency, modulation, symbol rate, and bit rate. Finally, by observing the output of the wake-up receiver with an oscilloscope, I was able to retrieve the identification number 0x7DA8. This confirmed that the receiver correctly processed incoming signals and extracted the expected data. The successful detection of this ID validated the proper configuration of the system, as shown in the oscilloscope reading in **Figure 10**.

Wow, incredible! But how long will a battery-powered electronic board last on a button cell? That's a great question. According to the manufacturer of RFicient, the lifespan of the wake-up receiver varies from 3 months to 11 years, depending on the data rate used. This is because power consumption is directly influenced by the chosen bit rate, which ranges from 64bps to 8,192bps. On the one hand, at higher data rates (closer to 8,192 bps), the receiver processes more data in a shorter time, leading to higher power consumption and a shorter battery life. On the other hand, at lower data rates (starting from 64bps), the system operates more efficiently, consuming minimal energy and significantly extending battery longevity.

The receiver I designed and developed is fully operational, with tests yielding positive results. Additionally, I had the opportunity to collaborate with the Fraunhofer University team in Germany, which allowed me to further refine my understanding of the receiver's startup specifications.

CONCLUSION

Traditional solutions involve periodic wake-up of the receiver, leading to significant power consumption for both the transmitter and the receiver. To address this challenge, within ALCIOM Research & Development company based in France, I developed a solution that reduces current consumption to just 1µA, while maintaining a sensitivity of -80dBm. This innovation significantly extends the battery life of connected devices, helping to reduce battery waste and minimize pollution.

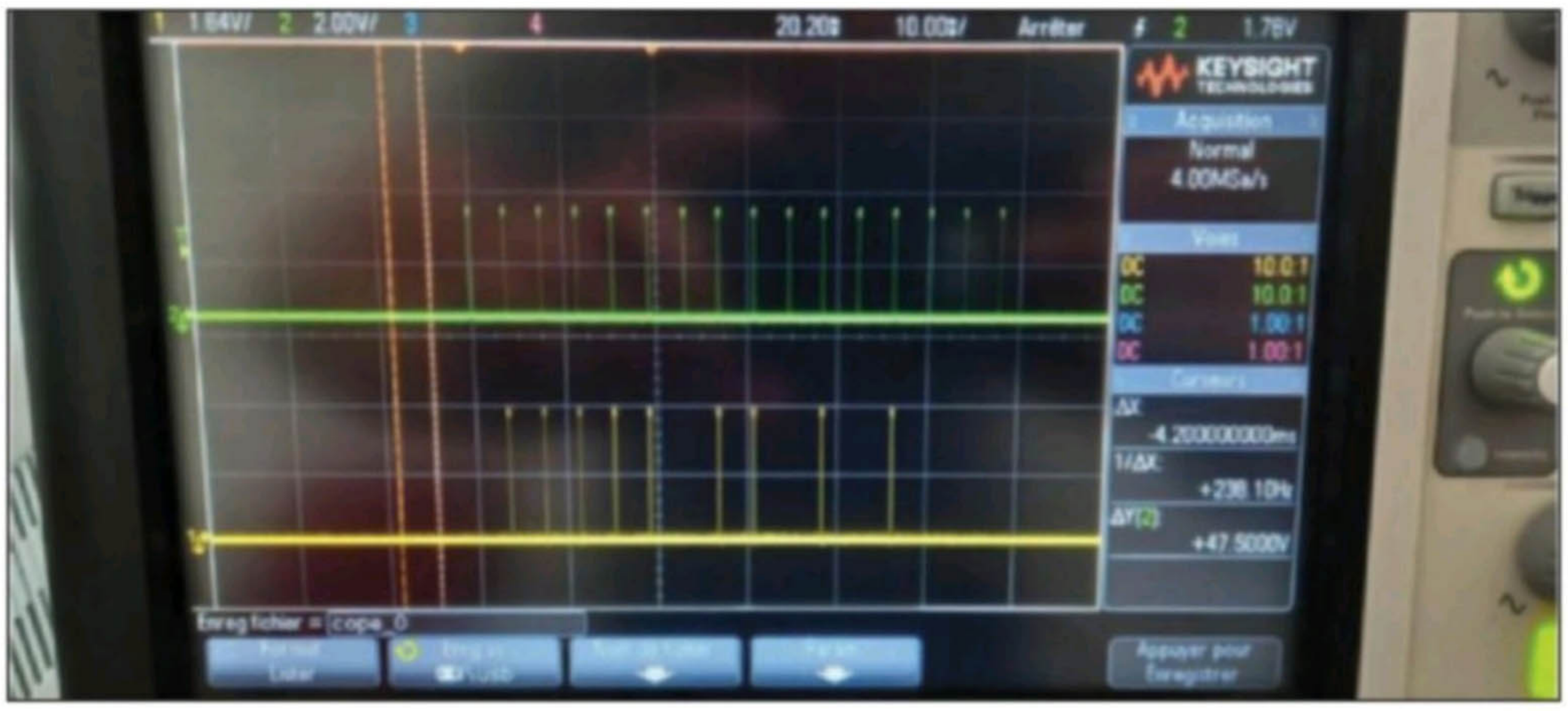
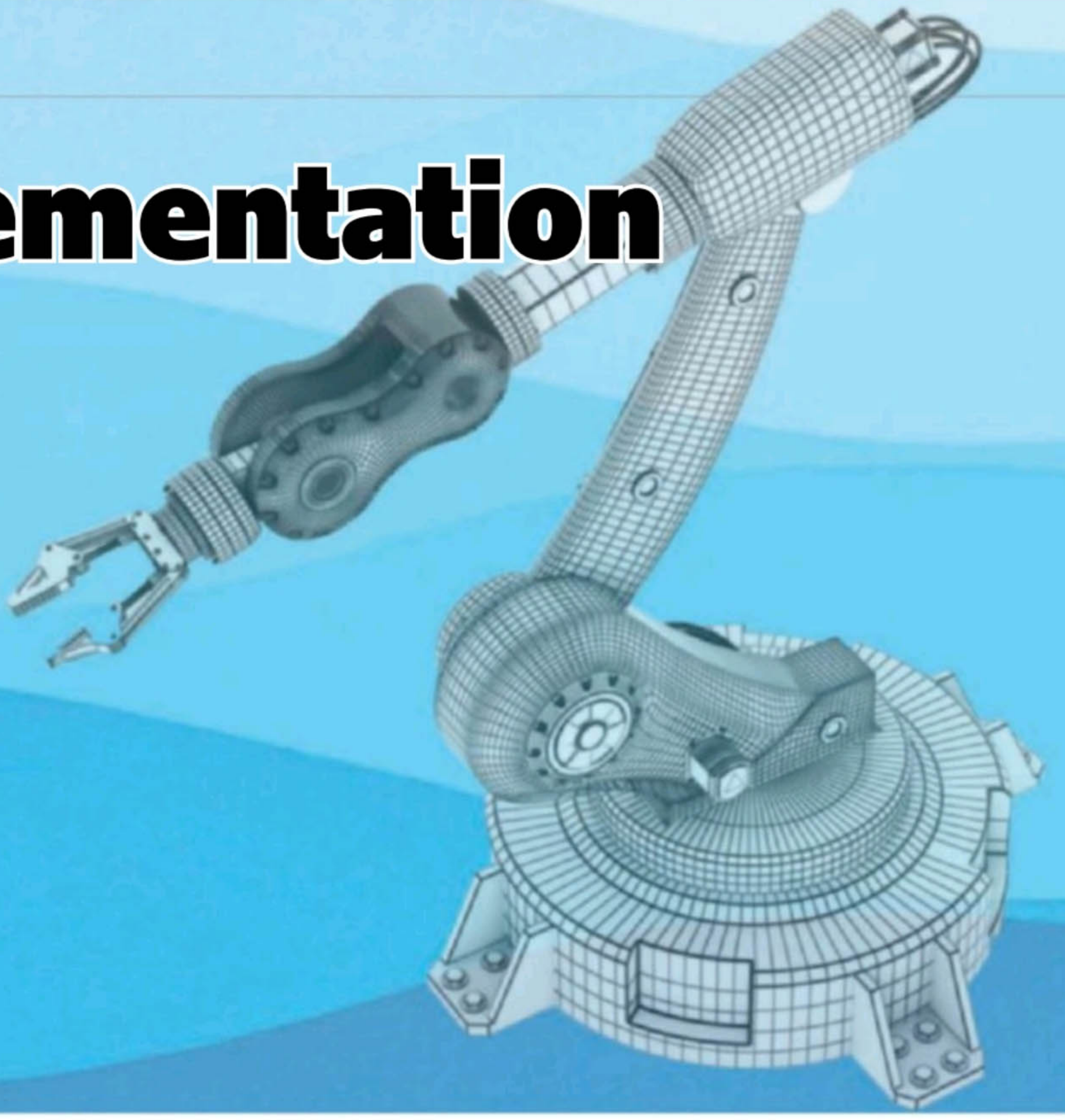


FIGURE 10
ID RFicient. An oscilloscope screen reading that validates the proper configuration of the system. (Source, ALCIOM [3])

ABOUT THE AUTHOR

Abel Didouh grew up in French Polynesia on the small island of Tahiti in the middle of the Pacific Ocean. Then he crossed two oceans and a continent to study in Paris, France. He studied mathematics and physics for 3 years, then moved into the field of electronics. He acquired a solid grounding in digital and analog electronics, and decided to specialize in the design, development, and production of radio systems. Now, he works as intern engineer in ALCIOM a consulting and design house company specializing in mixed signal electronics. Email: abel.didouh@edu.esiee.fr

Design and Implementation of a 3D-Printed Robotic Actuator



By
Maheshwaran Sadasivam and
Vairamani Kanagavel

A robotic actuator converts energy into physical motion, enabling a robot to move and interact with its environment. In this article, Maheshwaran and Vairamani describe how they designed and assembled a 3D-printed actuator that combines advanced motion control with real-time monitoring.

This article presents the design and implementation of a cost-effective, 3D-printed robotic actuator, specifically developed to provide students with an accessible and practical platform for learning and experimentation in robotics. The total expenses to make this 3D printed robotic actuator was $60, while the commercial robotic actuator was approximately $500. The actuator enables precise movement control and real-time monitoring for building robotic systems, such as robotic arms and quadruped robots (robot dogs).

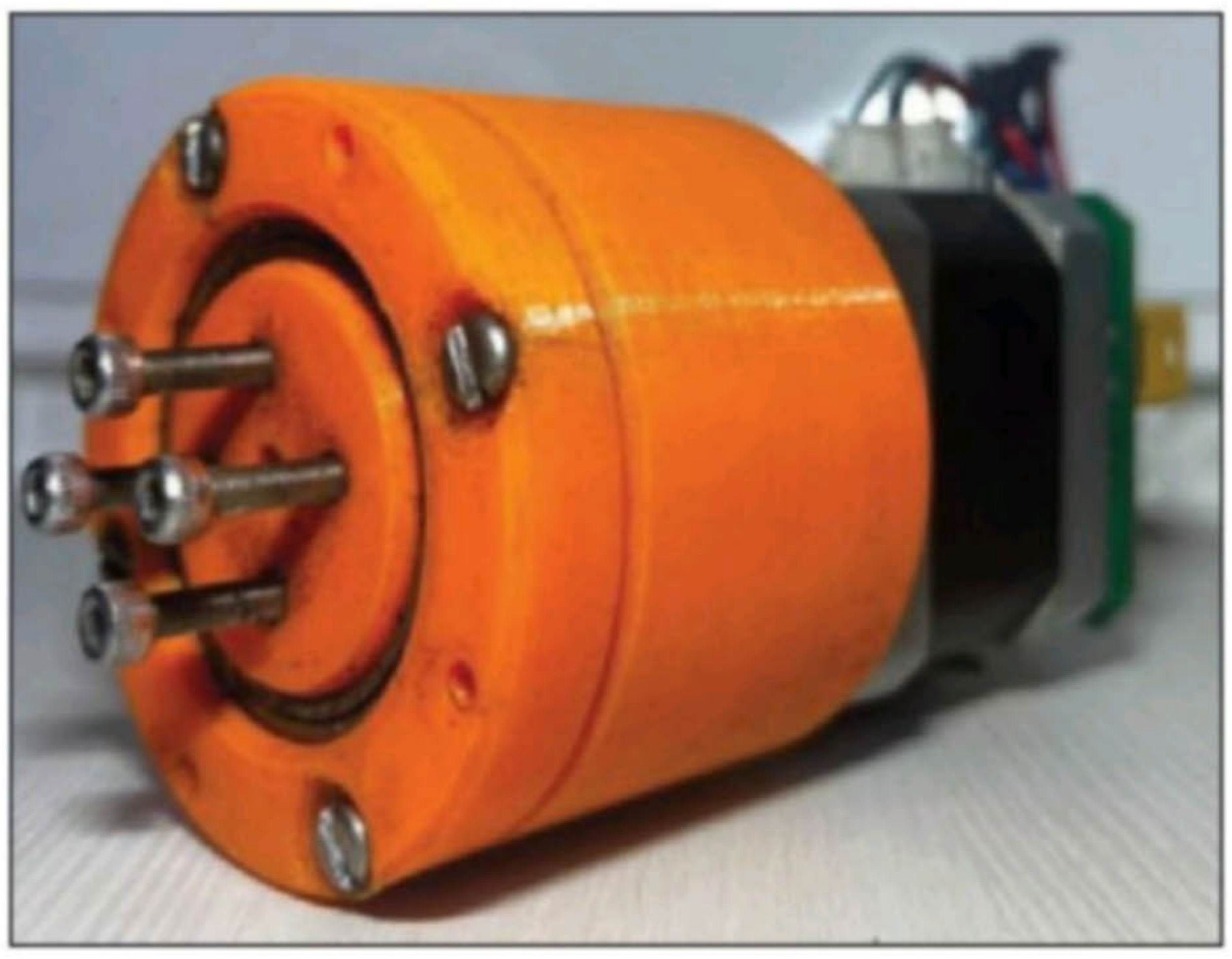


FIGURE 1
The 3D-printed robotic actuator enables precise movement control and real-time monitoring for building robotic systems.

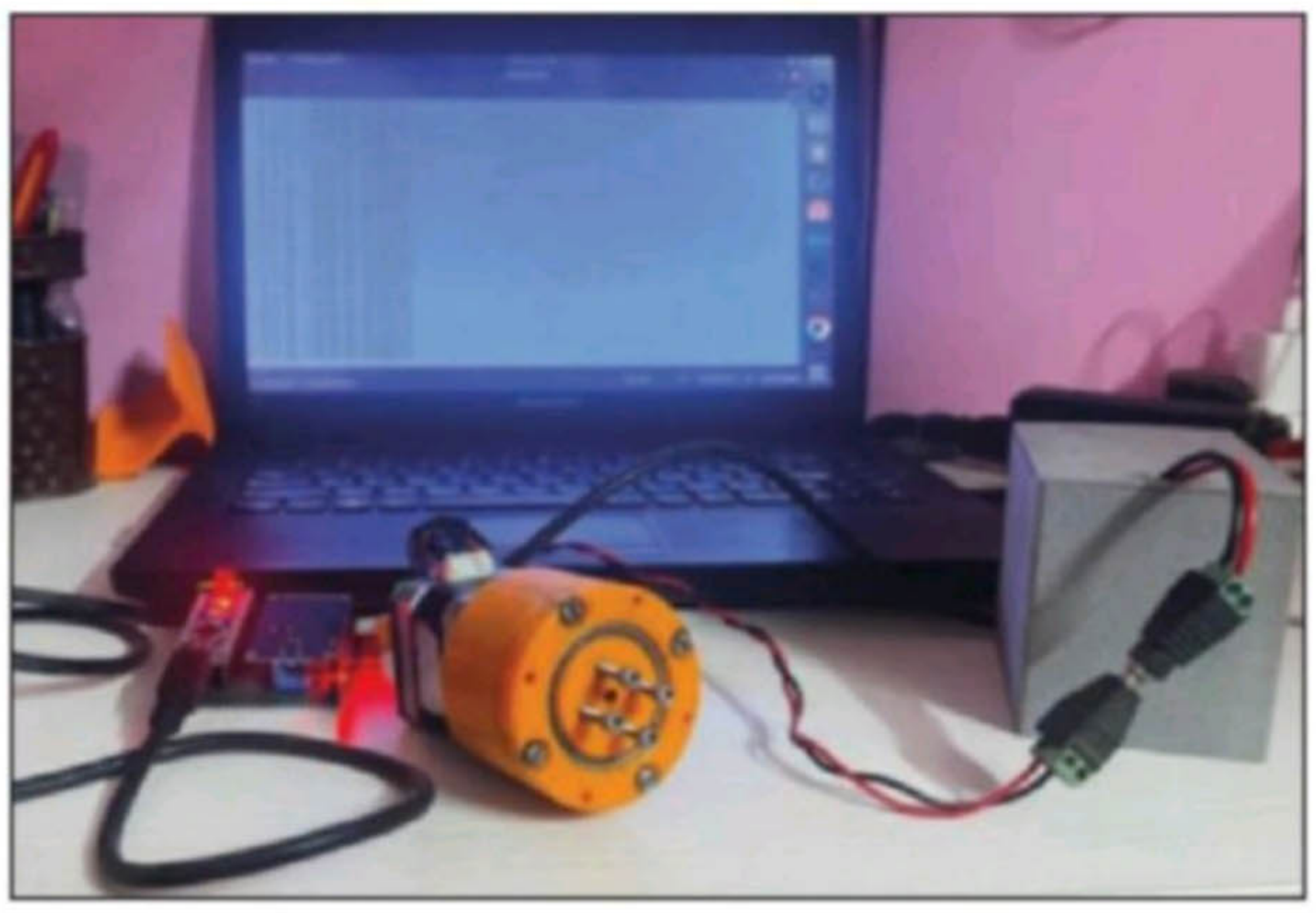


FIGURE 2
The actuator test setup of the 3D printed robotic actuator connected with 12V battery.

The actuator integrates a stepper motor with a 3D-printed planetary gearbox, providing efficient motion transmission. The overall system is controlled by a microcontroller, which interfaces with a magnetic encoder to offer feedback from the motor for closed-loop control, ensuring precise movement. Additionally, an inertial measurement unit (IMU) sensor measures the actuator's acceleration and orientation, allowing dynamic adjustments during operation. A current and voltage sensor monitors the actuator's energy consumption, optimizing performance and ensuring efficient power usage. Communication is facilitated via the controller area network (CAN) bus protocol, enabling reliable data exchange with external devices.

This versatile actuator's low-cost, scalable design makes it ideal for a range of robotic applications. Experimental results demonstrate its effectiveness in terms of positional accuracy, energy efficiency, and adaptability, highlighting its potential for use in both research and industrial robotics.

HARDWARE

The actuator is shown separately in **Figure 1**, and it is shown with the rest of the system in **Figure 2**, and a block diagram of the system is given in **Figure 3**. Next, we discuss each of the hardware components.

NEMA 17 Stepper Motor: The Twotrees 17HS4401 NEMA stepper motor (**Figure 4**) provides precise movement control in the robotic actuator. Known for its compact 1.7″ square faceplate, it offers a versatile range of torque and speed options, making it suitable for various robotic applications.

The motor operates on electromagnetic induction, where coils generate magnetic fields to drive the rotor in discrete steps. With a step angle of 1.8°, the NEMA 17 provides 200 steps per full rotation, ensuring high positional accuracy for the actuator.

Its relatively high torque-to-weight ratio and ability to hold position without continuous power make it an efficient choice for the actuator's design, contributing to reliable performance in dynamic environments. However, like all stepper motors, it may experience resonance at high speeds, which could affect performance under certain conditions.

Planetary Gearbox: An 80:1, 3D-printed planetary gearbox is used in conjunction with the NEMA 17 stepper motor to reduce speed and increase output torque. The 3D model of its parts is shown in **Figure 5**. The combination of the motor and gearbox is ideal for applications requiring high torque and precision within a compact, cost-effective package. I used PLA filament to print the planetary gearbox parts, for a balance among strength, ease of printing, and cost.

This gearbox functions by using "planet gears" that rotate around a central "sun gear," enabling efficient power transfer and load distribution. With the 80:1 reduction ratio, for every 80 rotations of the motor, the output shaft completes one full rotation, allowing the actuator to handle heavier loads without compromising performance.

This configuration ensures smoother operation, reduces speed, and enhances torque. Additionally, the design minimizes backlash, contributing to improved accuracy and reliable performance.

MPU6050 Accelerometer and Gyroscope Sensor: This equipment provides critical data on orientation, acceleration, and rotational velocity of the actuator (**Figure 6** and **Figure 7**). The MPU6050 combines a three-axis accelerometer and a three-axis gyroscope on a single chip, allowing it to measure both linear acceleration and rotational motion. The accelerometer tracks linear acceleration, while the gyroscope detects rotational movement, offering comprehensive motion data for precise control.

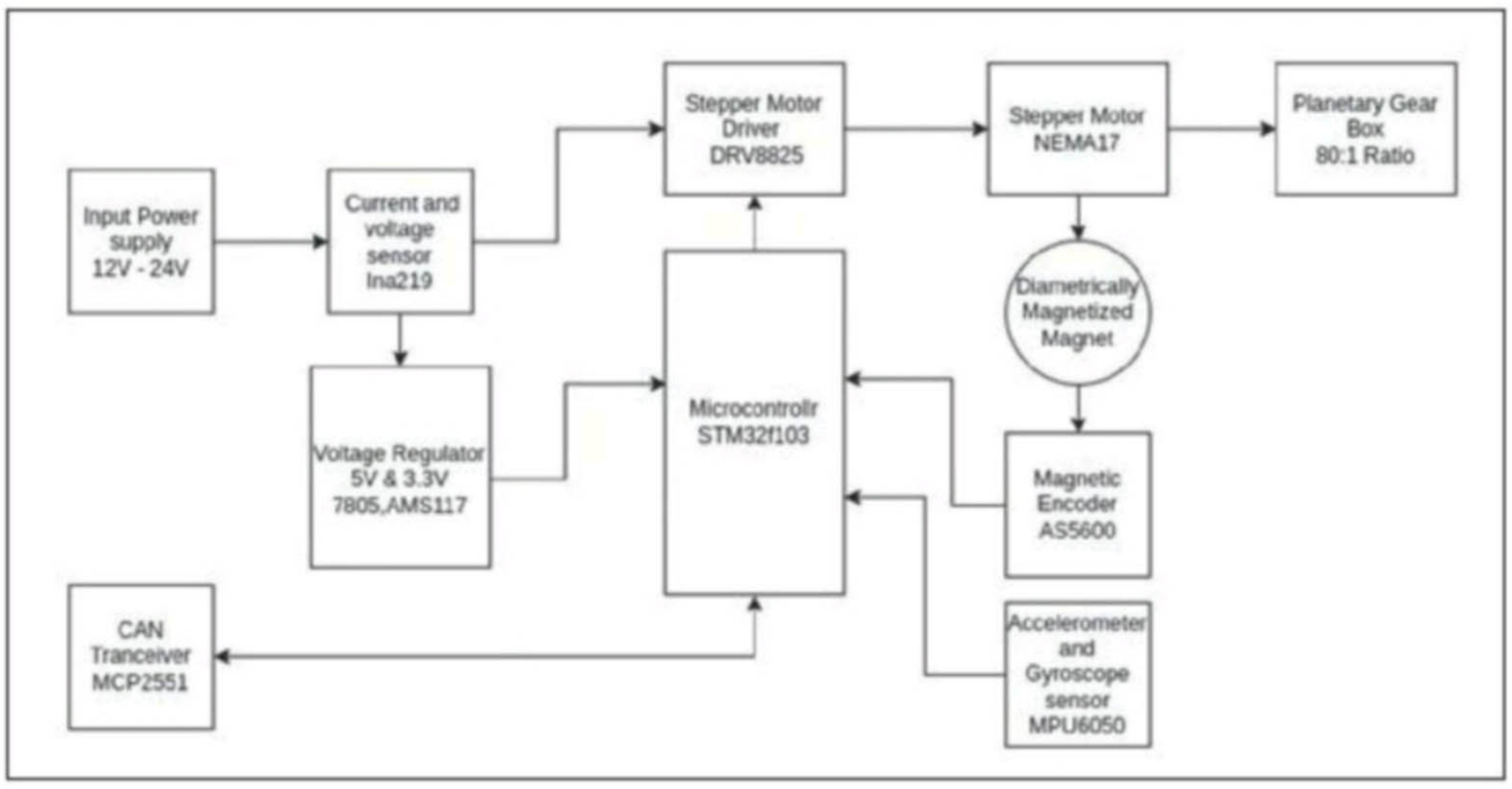

Input Power supply 12V - 24V
Current and voltage sensor Ina219
Voltage Regulator 5V & 3.3V 7805,AMS117
CAN Tranceiver MCP2551
Stepper Motor Driver DRV8825
Microcontrollr STM32f103
Stepper Motor NEMA17
Planetary Gear Box 80:1 Ratio
Diametrically Magnetized Magnet
Magnetic Encoder AS5600
Accelerometer and Gyroscope sensor MPU6050

FIGURE 3
Block diagram of the actuator system.

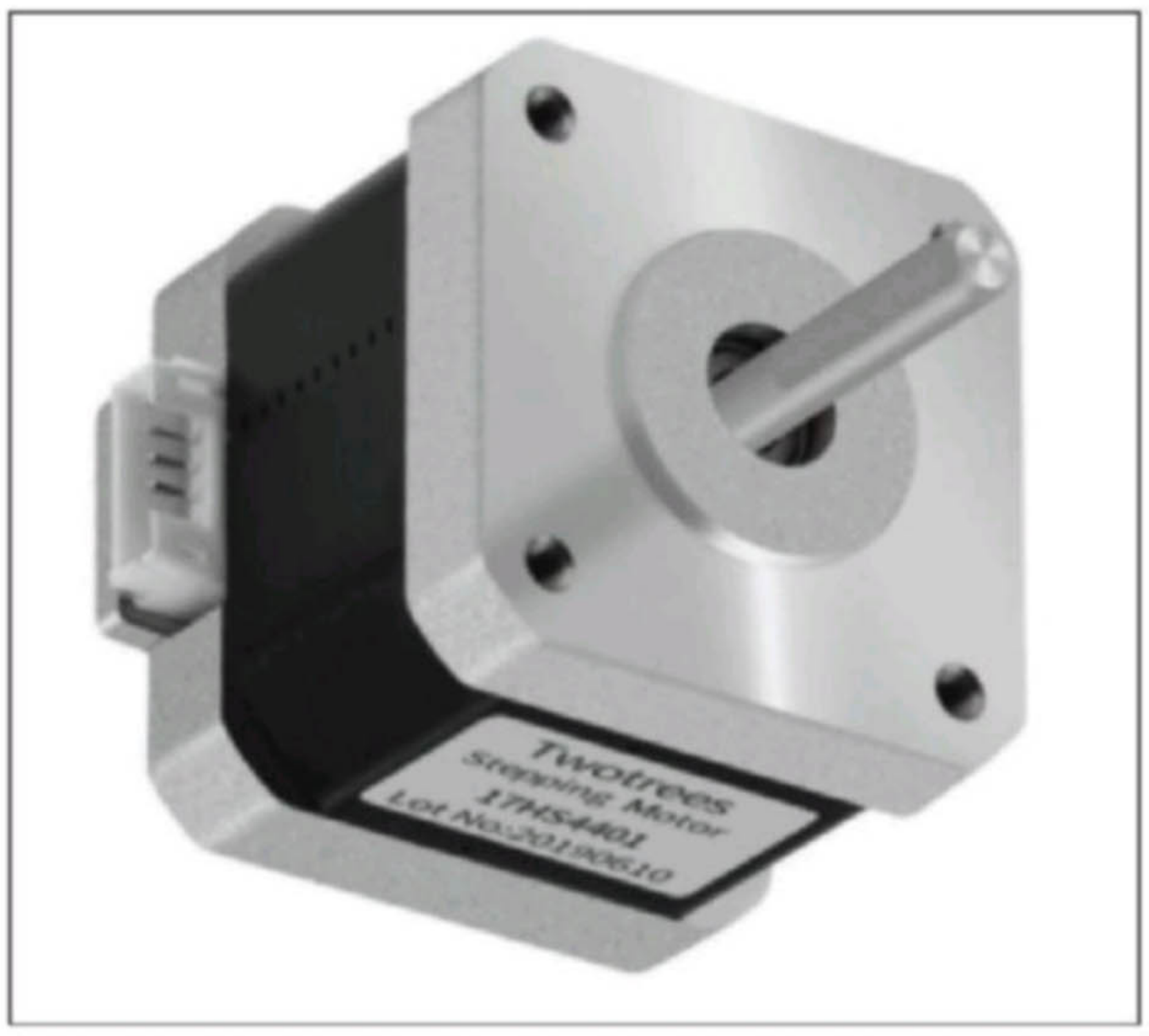

Twotrees
Stepping Motor
17HS4401
Lot No:20190610

FIGURE 4
NEMA 17 Stepper Motor with a compact 1.7″ square faceplate, offers a versatile range of torque and speed options.

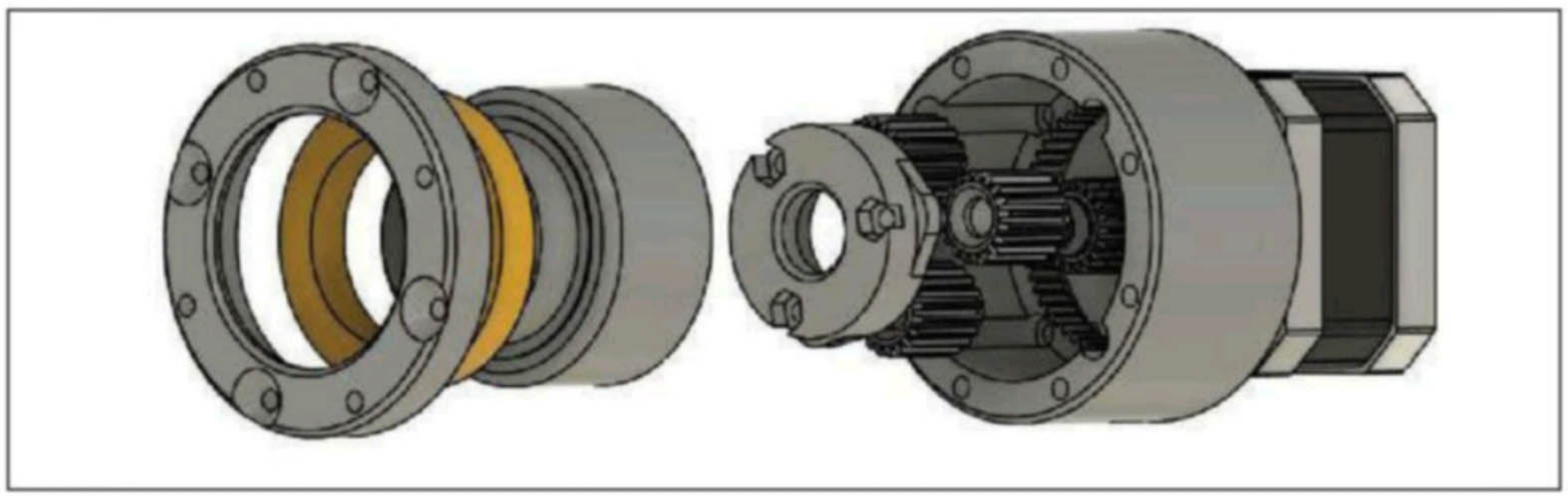


FIGURE 5
3D model of planetary gearbox parts. The parts are used in conjunction with the NEMA 17 stepper motor to reduce speed and increase output torque.

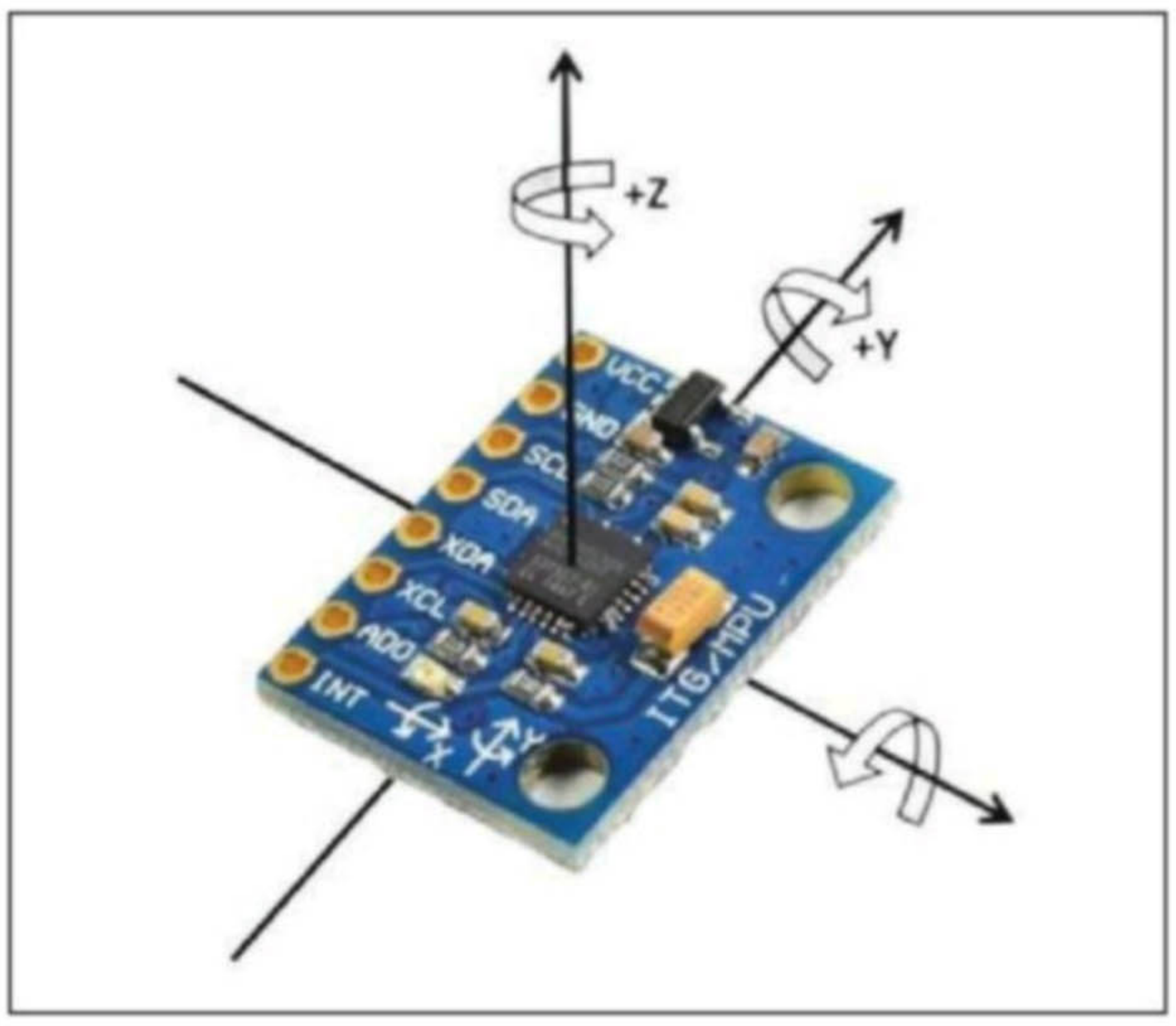

+Z
+Y
VCC
GND
SCL
SDA
XDA
XCL
AD0
INT
ITG/MPU
X
Y

FIGURE 6
The MPU6050 IMU sensor, MPU6050 accelerometer, and gyroscope sensor.

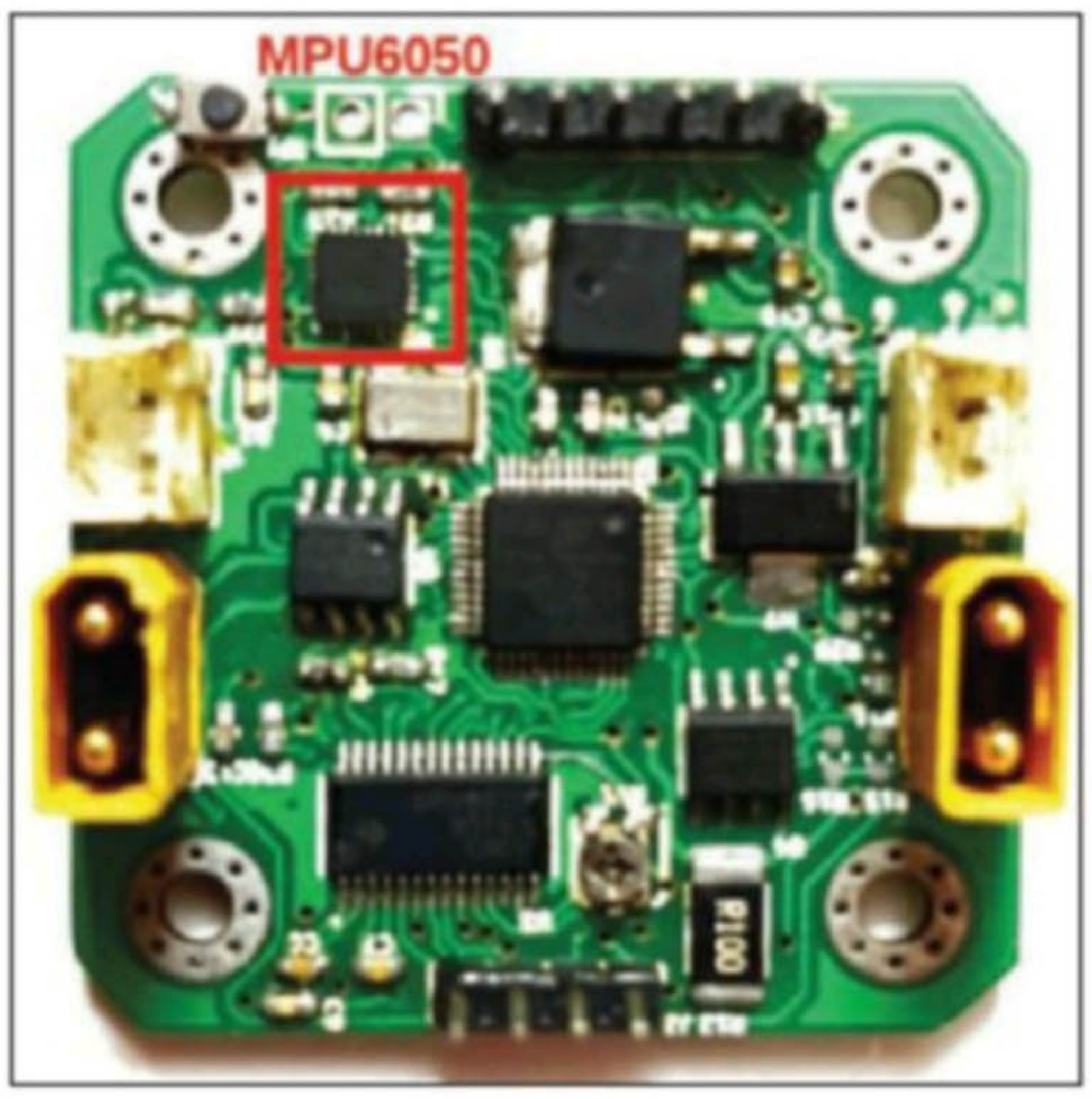


FIGURE 7
The MPU6050 accelerometer and gyroscope sensor installed on the circuit board.

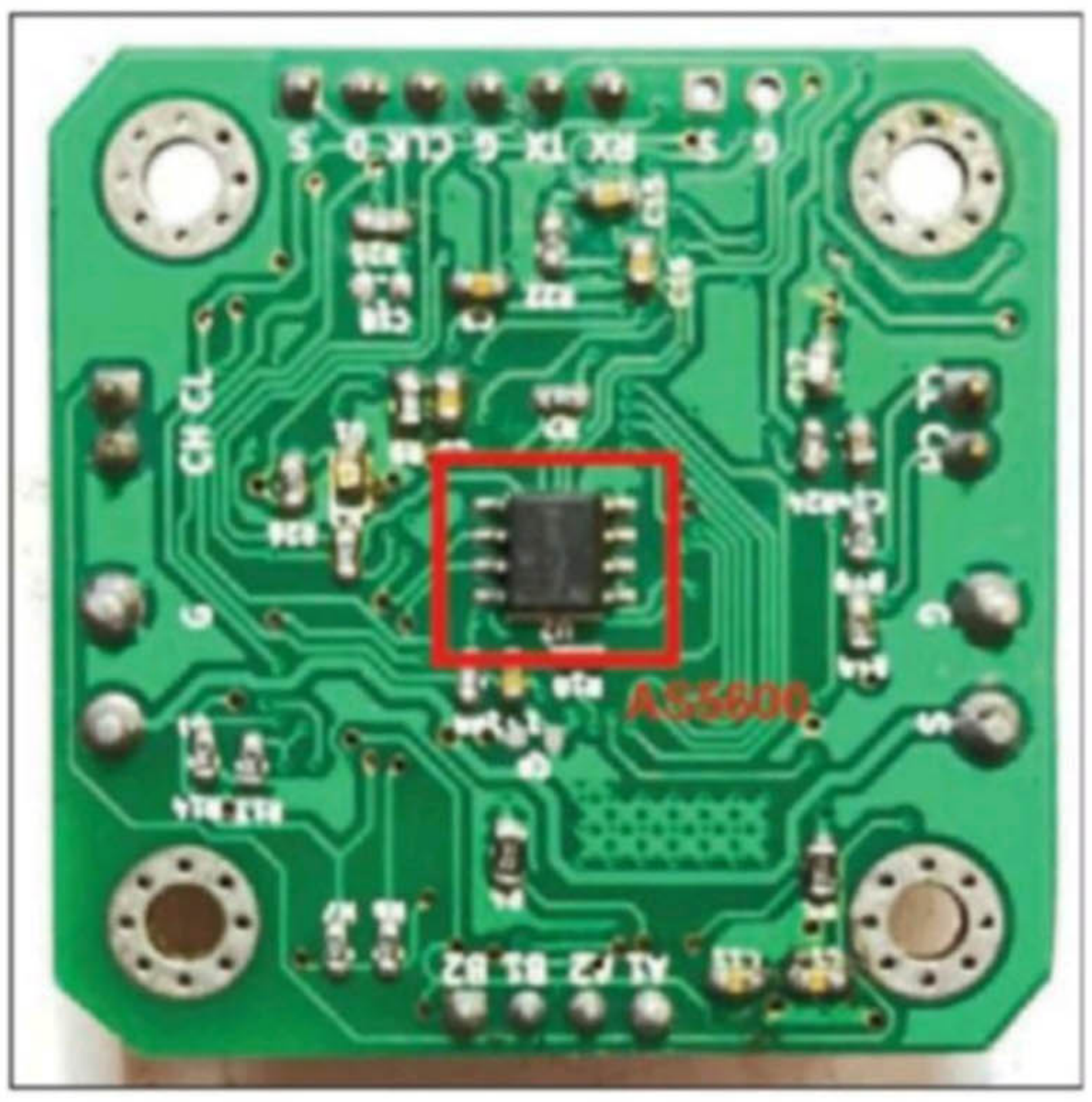


FIGURE 8
The AS5600 Magnetic Encoder installed on the backside of the circuit board.

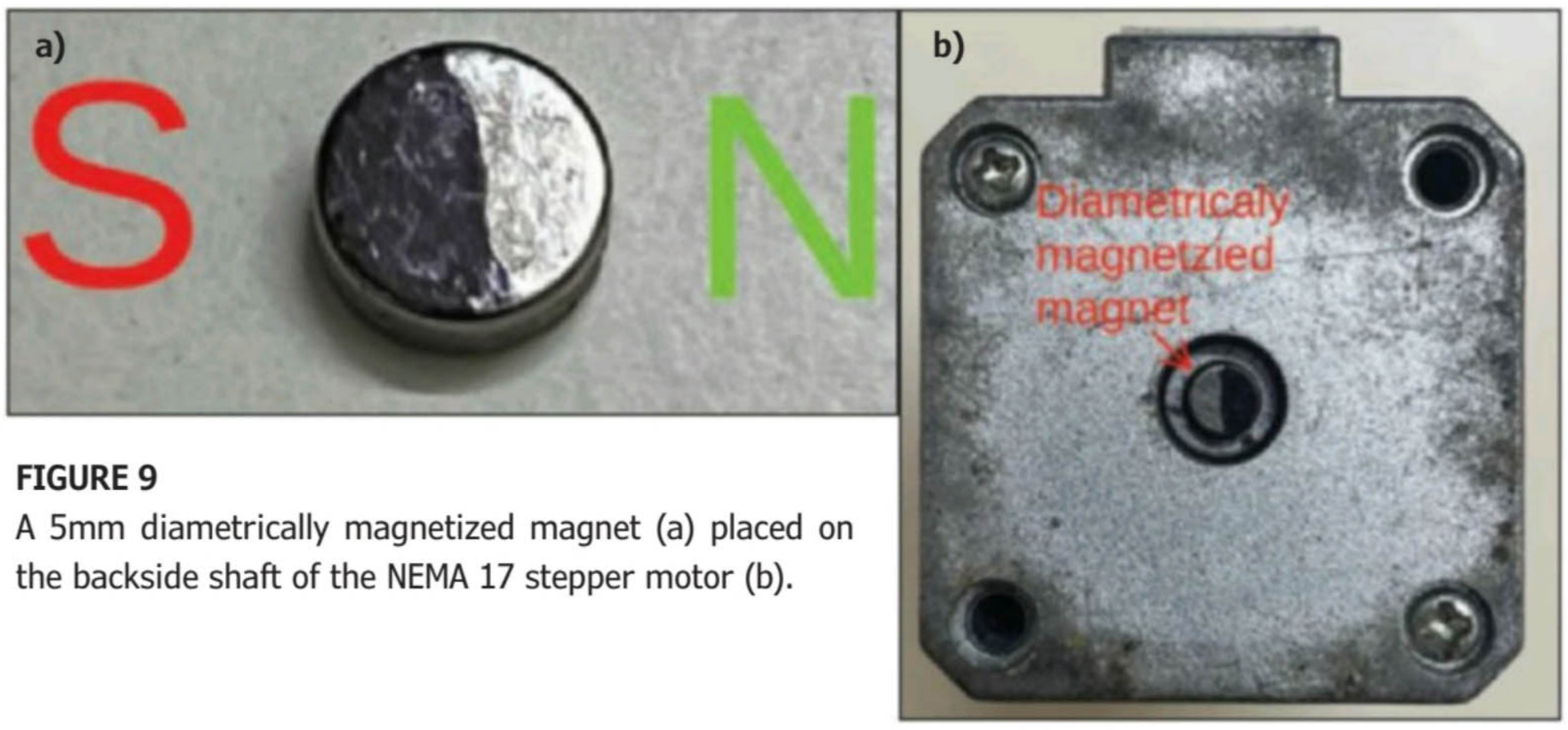


FIGURE 9
A 5mm diametrically magnetized magnet (a) placed on the backside shaft of the NEMA 17 stepper motor (b).

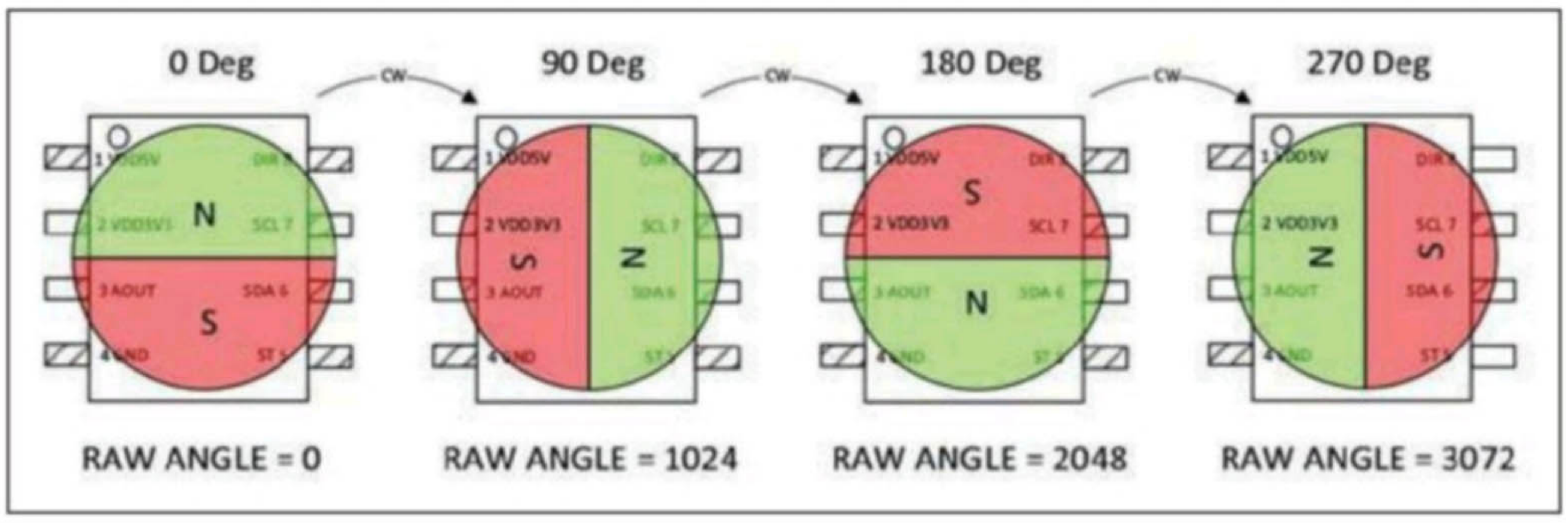


FIGURE 10
Illustration showing how the AS5600 interacts with the diametrically magnetized magnet.

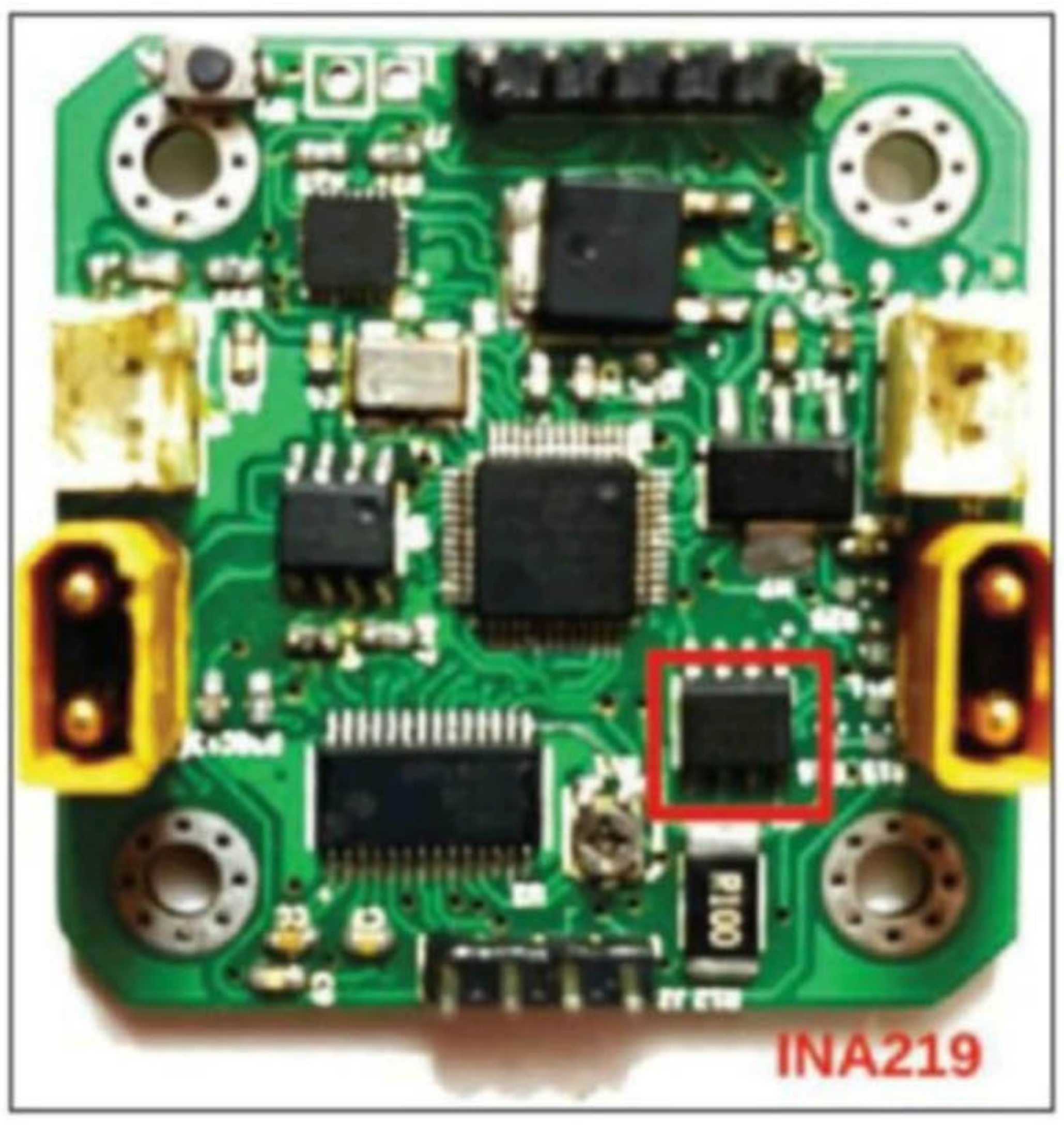


FIGURE 11
The INA219 current and voltage sensor installed on the circuit board.

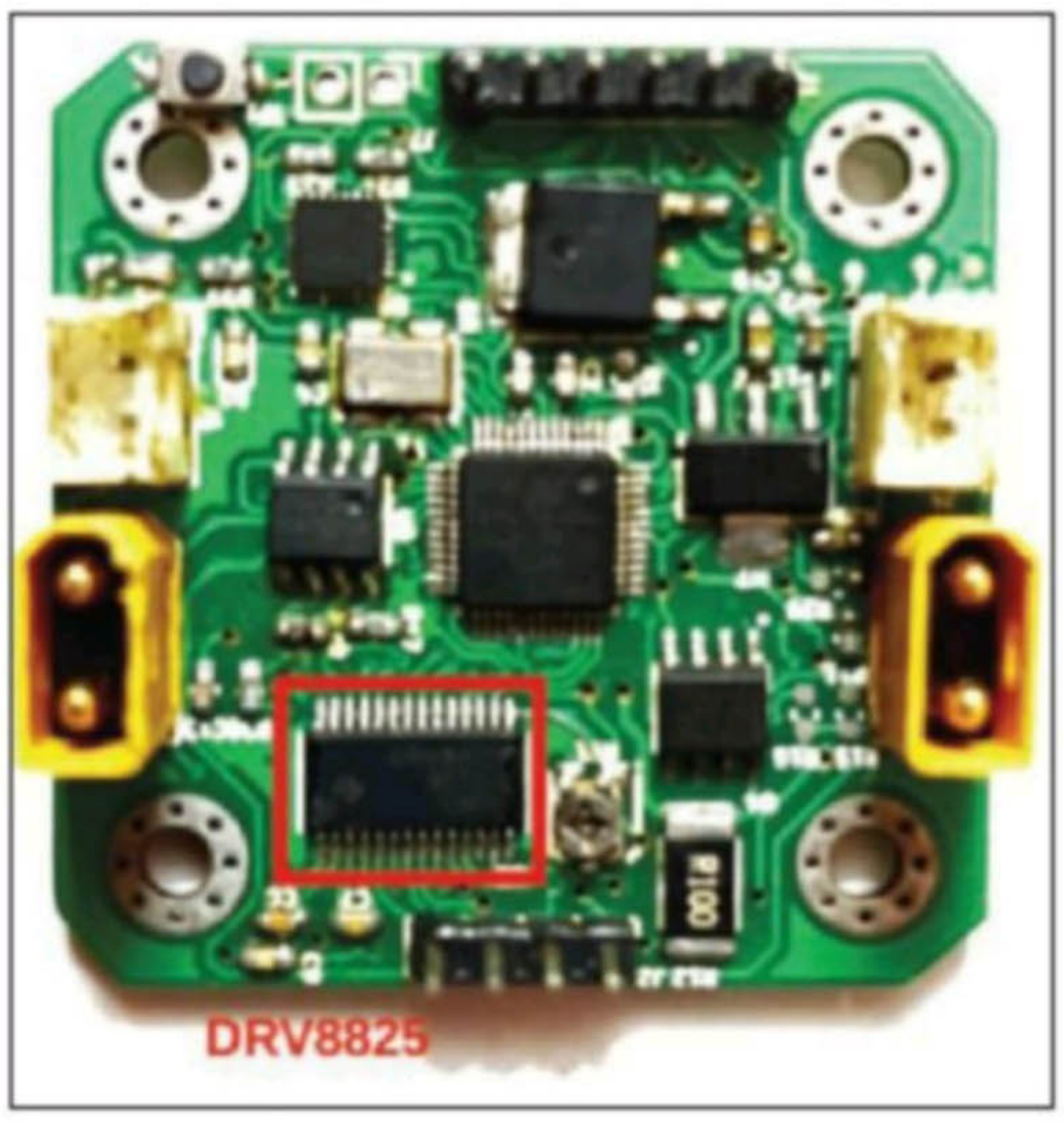


FIGURE 12
The Texas Instruments DRV8825 stepper motor driver installed on the circuit board.

The chip communicates with the microcontroller using the I²C protocol, enabling efficient data exchange. This real-time feedback on the actuator's orientation and motion allows for dynamic adjustments during operation. By integrating the MPU6050, the actuator achieves enhanced stability and responsiveness, making it well-suited for applications requiring precise movement control. The sensor's ability to process complex motion data through its built-in Digital Motion Processor (DMP) also reduces the load on the microcontroller, improving overall system efficiency and performance.

Magnetic Encoder: The AS5600 magnetic rotary position sensor (**Figure 8**) provides precise angular position feedback. Utilizing Hall-effect technology, it provides accurate and reliable position data, ensuring smooth and controlled movement. With a resolution of up to 12 bits, it delivers the high precision needed for tasks that require fine positioning.

To measure the angle, a diametrically magnetized magnet (**Figure 9a**) is placed on the back side of the NEMA 17 stepper motor shaft (**Figure 9b**). This positioning allows the AS5600 to detect changes in the magnetic field as the shaft rotates, resulting in precise angle measurements. The AS5600 sensor specifically detects the shifts between the North and South poles of the diametrically magnetized magnet as it rotates. These pole changes are translated into angular positions, providing accurate measurements (**Figure 10**). The clear line of sight between the sensor and the magnet minimizes interference and enhances the sensor's accuracy, ensuring reliable readings throughout the rotation of the motor shaft. The speed at which these changes happen depends on the motor's revolutions per minute (rpm). As the shaft rotates, the AS5600 detects the magnetic field changes in real time. For each revolution, it registers a full cycle of magnetic pole shifts, allowing it to track the rpm accurately. Since the AS5600 has a high resolution, the faster the motor shaft rotates, the more frequent the changes in magnetic field will be detected, allowing the sensor to calculate the rpm precisely

The AS5600 supports both I²C and PWM outputs, facilitating easy integration with various microcontrollers and robotic systems. Its low power consumption and robust design make it suitable for mobile robots, where efficiency and durability are paramount. Additionally, the sensor features a programmable zero position and operates effectively across a wide temperature range, ensuring consistent performance in diverse environments.

Overall, the AS5600 enhances the accuracy and reliability of the robotic actuator, making it essential for achieving precise control.

***Current and Voltage Sensor*:** The INA219 current and voltage sensor (**Figure 11**) monitors the power consumption of the entire actuator system. By measuring the voltage across a shunt resistor and monitoring current, it provides real-time data to optimize energy usage and improve system efficiency. The IC communicates via I²C, making it easy to interface with popular microcontrollers such as Arduino or STM32.

Using the INA219, robotic systems can optimize actuator performance, track energy usage, and improve overall efficiency. This data is crucial for enhancing battery life, debugging issues, and ensuring the reliable operation of robots in various tasks.

Stepper Motor Driver: The Texas Instruments (TI) DRV8825 stepper motor driver (**Figure 12**) is particularly well suited for controlling robot actuators. Its key features include:

- Voltage Range: Operates from 8.2V to 45V.
- Current Handling: Supports up to 2.5A per phase with proper heatsinking.
- Microstepping: The stepper driver offers adjustable microstepping from full-step to 1/32-step, allowing for smoother operation and finer control over motor positioning.
- Built-In Protection: Features thermal shutdown, undervoltage lockout, and current limiting to safeguard both the driver and the motor.

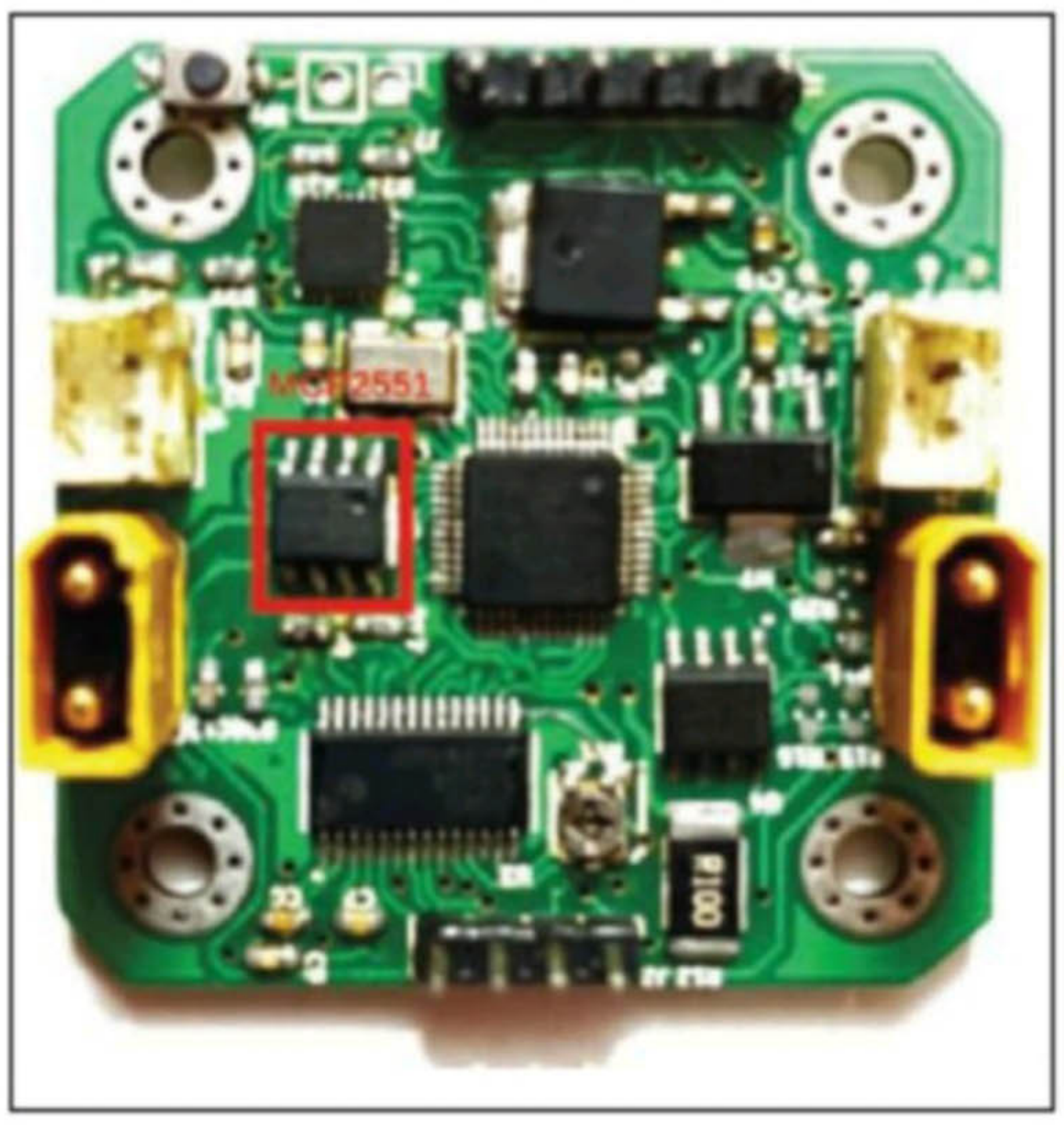


FIGURE 13
The MCP2551 CAN transceiver installed on the circuit board.

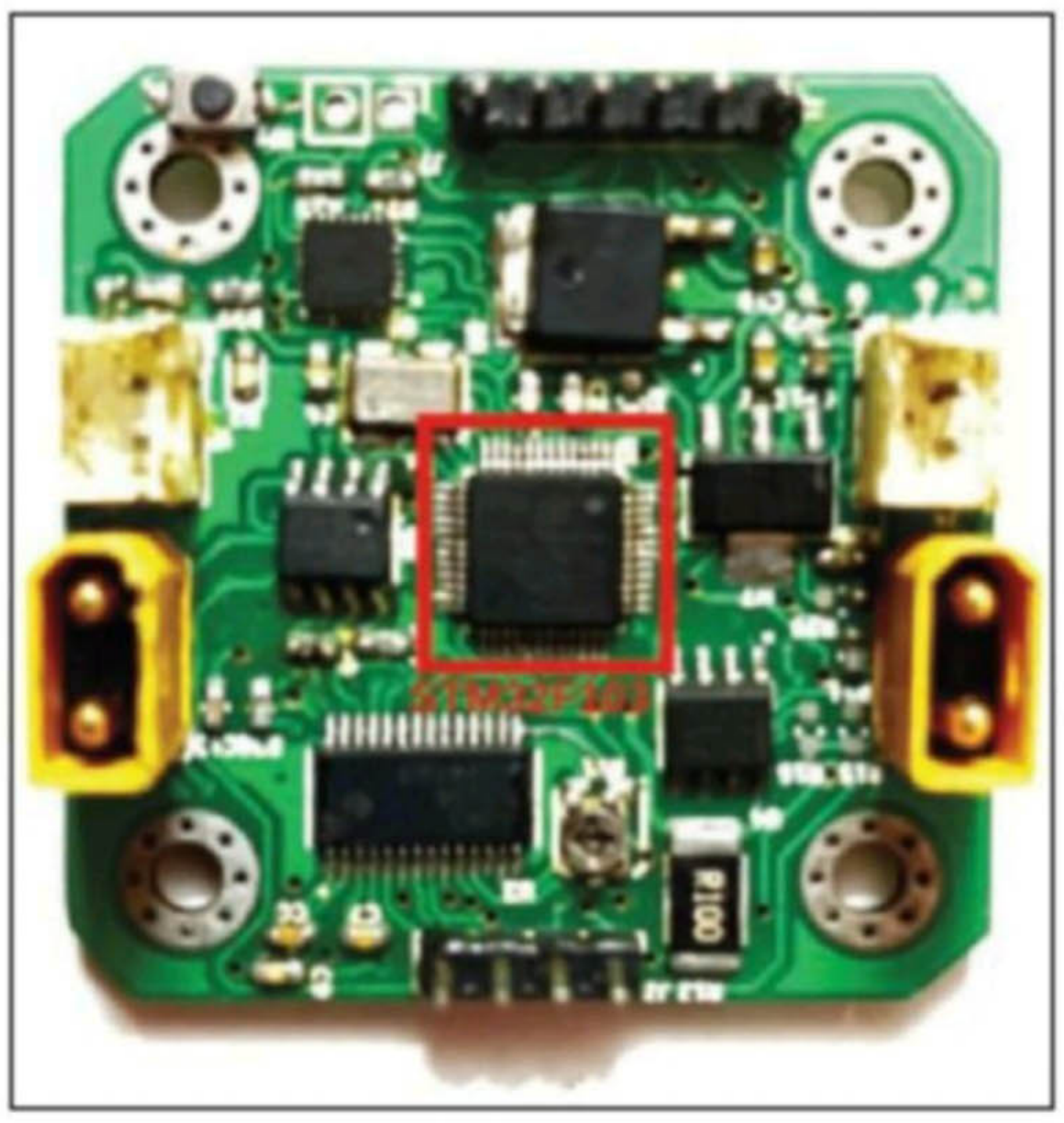


FIGURE 14
The STM32F103 microcontroller installed on the circuit board .

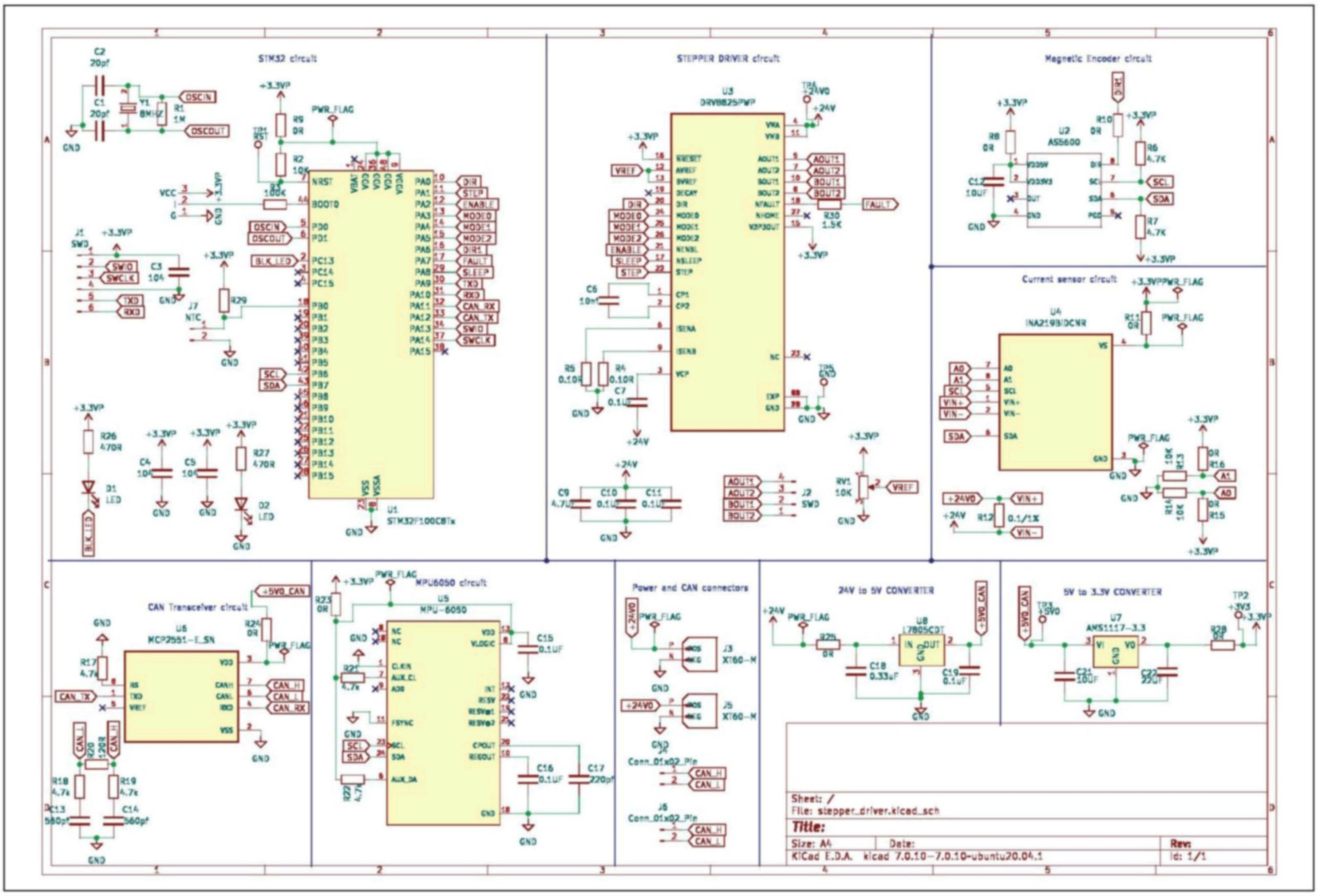

STM32 circuit
STM32F100CBTx
STEPPER DRIVER circuit
DRV8825PWP
Magnetic Encoder circuit
AS5600
Current sensor circuit
INA219BIDCNR
CAN Transceiver circuit
MCP2551-E_SN
MPU6050 circuit
MPU-6050
Power and CAN connectors
XT60-M
24V to 5V CONVERTER
L7805CDT
5V to 3.3V CONVERTER
AMS1117-3.3
Sheet: /
File: stepper_driver.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 7.0.10-7.0.10-ubuntu20.04.1
Id: 1/1

FIGURE 15
This circuit diagram shows the microcontroller, stepper driver, magnetic encoder, current sensor, CAN transceiver, IMU sensor, and power converters.

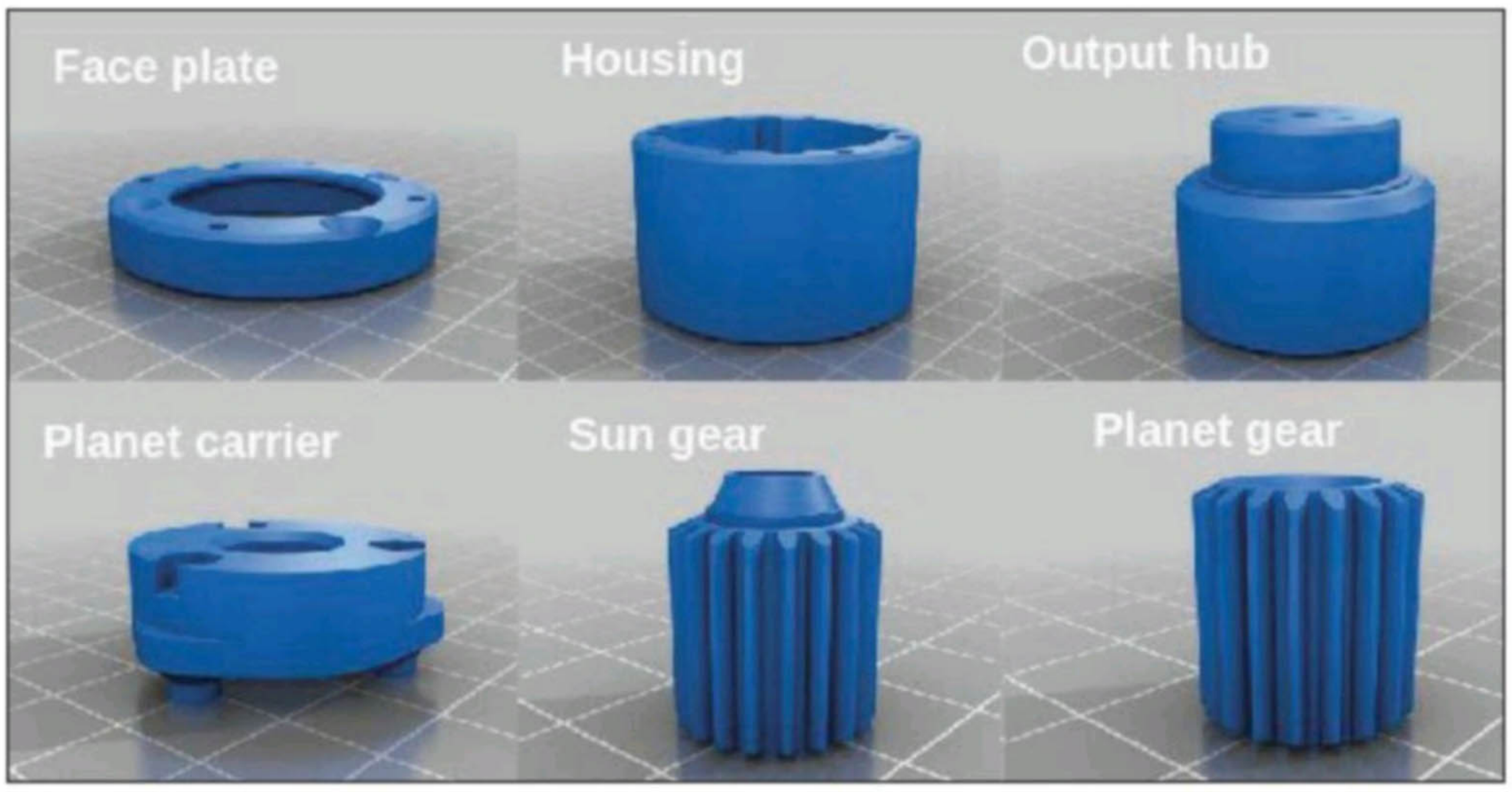


FIGURE 16
These are the 3D-printed planet gearbox parts. Be sure to make three planet gears.

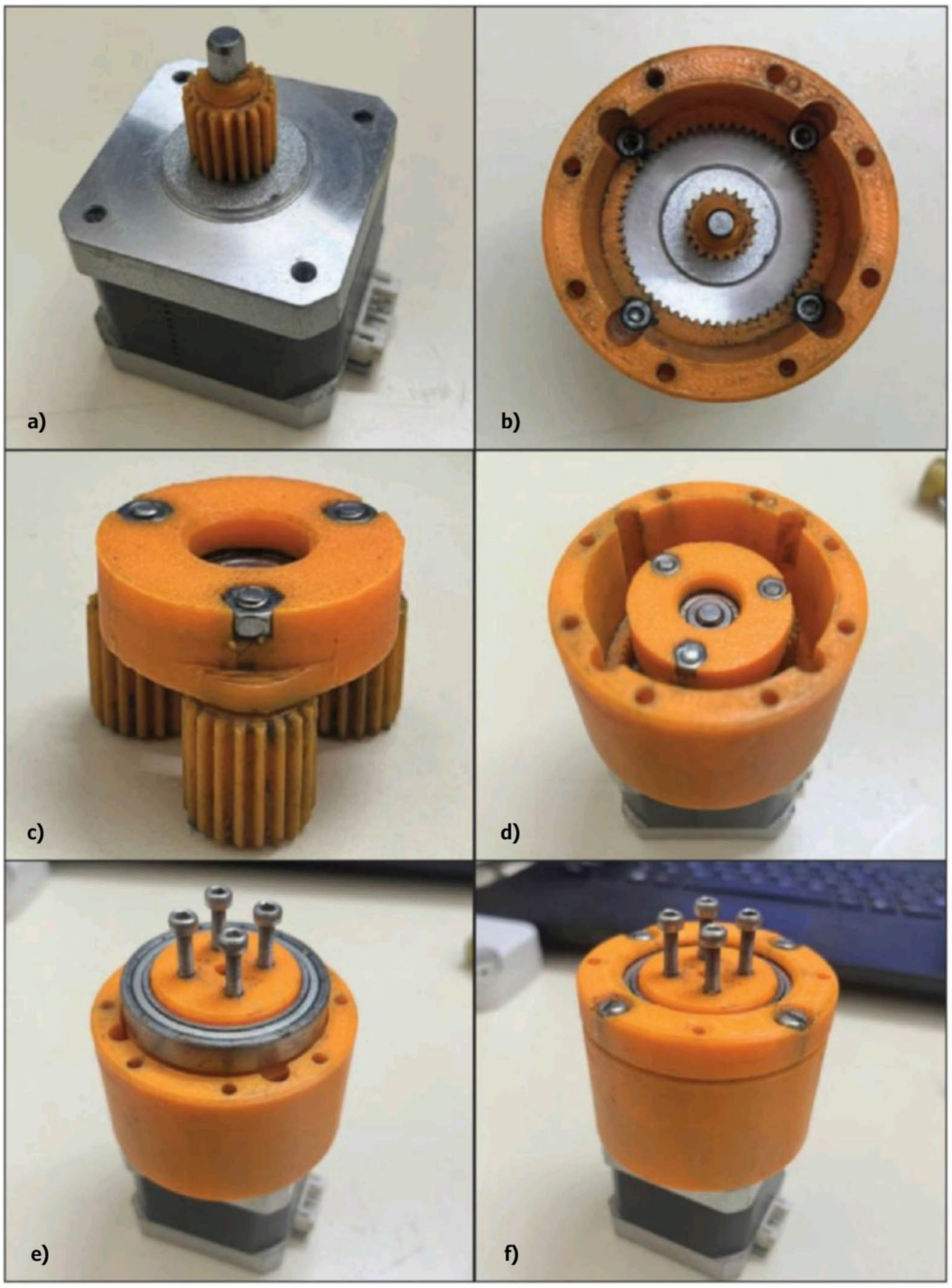


FIGURE 17
Sun gear attached with D-shaft NEMA 17 stepper motor (a). Housing attached with NEMA 17 stepper motor using M3 x 10mm allen socket head screws (b). Planet gears attached with planet carrier (c). Assembled planet carrier attached with housing unit (d). Output hub attached with housing unit (e). The completly assembled 80:1 planetary gearbox attached with NEMA 7 stepper motor (f).

CAN Transceiver: The MCP2551 (**Figure 13**) is a high-speed CAN transceiver that facilitates communication between various components in robotic systems, particularly those using robotic actuators. Designed to meet the demands of CAN applications, its key features include:

- High-Speed Communication: Supports data rates up to 1Mbps, allowing for fast and efficient communication between the microcontroller and multiple actuators.
- Signal Integrity: Built-in features that enhance noise immunity ensure reliable operation even in electrically noisy environments, which are common in robotic applications.
- Ease of Integration: Compatible with standard microcontrollers, making it easy to integrate into existing robotic systems. Its compact design and simple pin configuration simplify the layout in compact robotic platforms.
- Wide Operating Range: Operates across a wide voltage range (4.5V to 5.5V), providing flexibility for various power supply configurations commonly used in robotics.
- Thermal Performance: Can operate effectively over a wide temperature range, which is critical for outdoor or harsh environment applications.

In robotic systems, actuators are essential for movement and control. The MCP2551 enables seamless communication between the central processing unit (CPU) and the actuator, ensuring synchronized and responsive operation.

Microcontroller: The STMicroelectronics STM32F103 microcontroller (**Figure 14**) is a 32-bit ARM Cortex-M3-based MCU known for its balance of performance, power efficiency, and rich feature set, making it an excellent choice for controlling robotic actuators. Its robotics applications include:

- Control Actuators: Precise control of motors allows for smooth and accurate movement.
- Implement Feedback Loops: By integrating sensors (encoders, gyroscopes), it processes feedback to adjust actuator positions dynamically, enhancing responsiveness and accuracy.
- Enable Autonomous Functionality: Can handle complex algorithms for navigation, obstacle detection, and decision-making, making it suitable for autonomous robots.

Figure 15 is a circuit diagram showing the microcontroller, stepper driver, magnetic encoder, current sensor, CAN transceiver, IMU sensor, and power converters. The next sections of this article contain instructions for assembling the actuator and related components.

3D PRINTING AND ASSEMBLY OF PLANETARY GEARBOX

1) Download the STL files for the 80:1 planetary gearbox, which includes the faceplate, housing, output hub, planet carrier, sun gear, and planet gears. Remember to print three planet gears. See **Figure 16** for reference. Print the parts with a 3D printer, using PLA or PETG filament.
2) Insert the sun gear onto the D-shaped shaft of the NEMA 17 stepper motor (**Figure 17a**).
3) Attach the housing to the NEMA 17 stepper motor using four M3 x 10mm Allen socket head screws (**Figure 17b**).
4) Insert the 625-2RS ball bearing into the bearing slot at the center of the planet carrier, ensuring a proper fit.
5) Install the 683ZZ ball bearings into each of the three planet gears, one at a time. Then, place the planet gears (into the slots of the planet carrier, securing them with three M3 x 14mm Allen socket head screws and M3 hex nuts (**Figure 17c**).
6) Insert the planet carrier assembly (with the planet gears) into the housing (**Figure 17d**).
7) Place the four M3 hex nuts into the inner slots of the output hub.
8) Apply a layer of lubrication, such as grease, to ensure smooth operation.
9) Insert the output hub assembly into the housing, ensuring a secure fit (**Figure 17e**).
10) Insert the 6806-2RS ball bearing into the slot on the faceplate.
11) Insert the faceplate assembly into the housing and secure it using the four M3 x 14mm Allen socket head screws.
12) Use four M3 x 14mm Allen socket head screws to attach the load to the output hub of the actuator (**Figure 17f**).

ASSEMBLY OF THE PCB

1) Download the Gerber file for the robotic actuator and order the PCB from JLCPCB. See **Figure 18** for reference.
2) Assemble the surface mount device components. You can either solder each component by hand one by one, or use a stencil and reflow oven for assembly.
3) Solder the through-hole components.
4) Once soldering is complete, verify that all components are seated properly and ensure there are no shorts.

ASSEMBLY OF PCB WITH ACTUATOR

1) Remove two M3 x 30mm screws from the NEMA 17 stepper motor diagonally.
2) Insert the 5mm magnet, magnetized diametrically, into the backside of the NEMA 17 shaft (Figure 9b).
3) Fix the PCB to the back side of the NEMA 17 stepper motor, using two M3 x 35mm Allen socket head screws, adding one M3 hex nut to each screw. The M3 hex nuts will act as spacers (**Figure 19a**).
4) This will lift the PCB 2.5mm away from the stepper motor's back side, helping prevent any short circuits on the PCB components (**Figure 19b**).
5) Connect the 4-pin JST connector cable from the NEMA 17 stepper motor to the PCB's 4-pin Bergstick header.
6) Connect the battery or power supply with an XT30 connector to the PCB to power the actuator.

UPLOADING FIRMWARE TO THE ACTUATOR

1) Install the STM32 in Arduino IDE.
    - Open Arduino IDE.
    - Go to File << Preferences.
    - In the Additional Boards Manager URLs field, add:

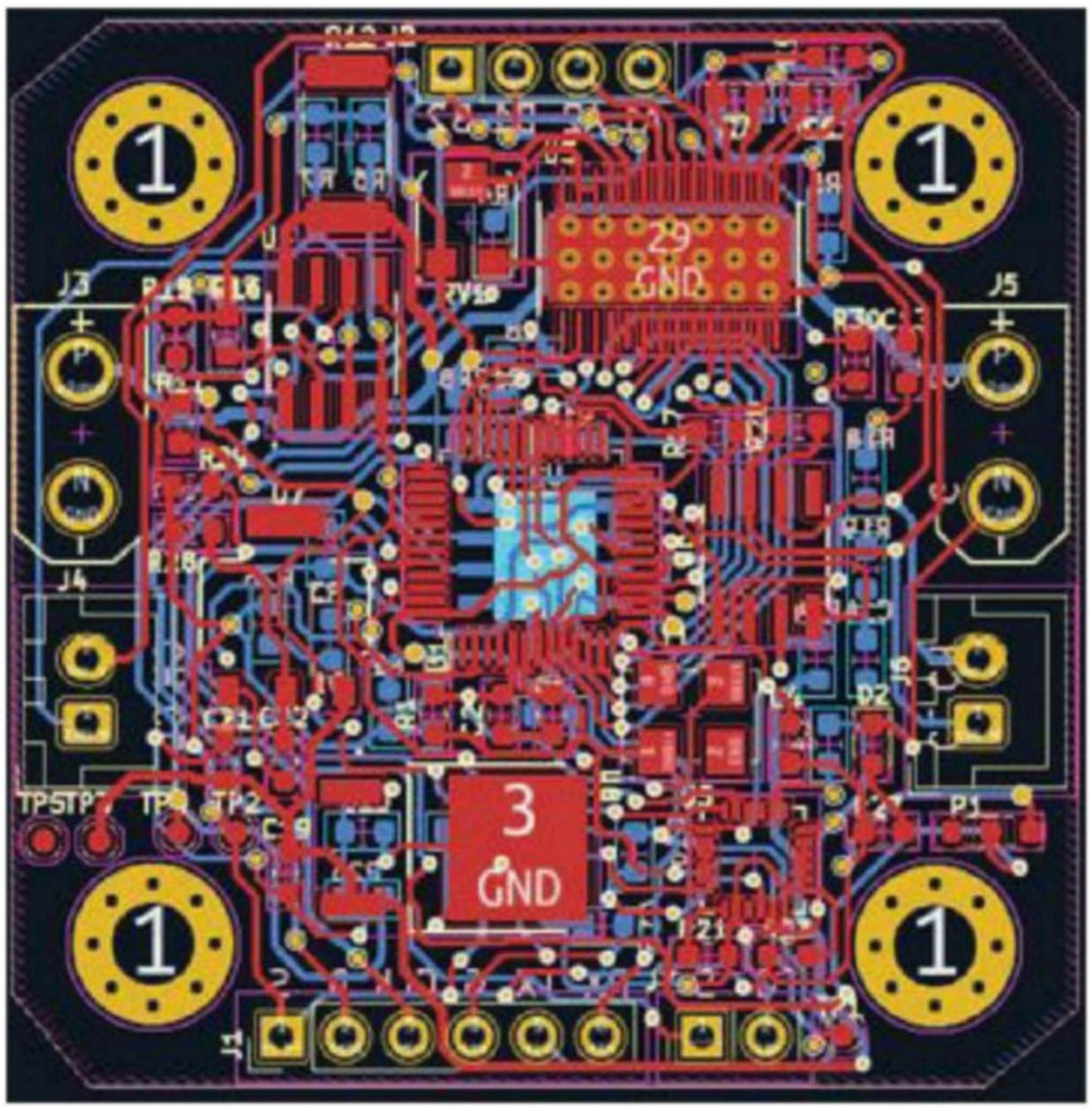


FIGURE 18
PCB layout of 3D-printed Robotic Actuator circuit.

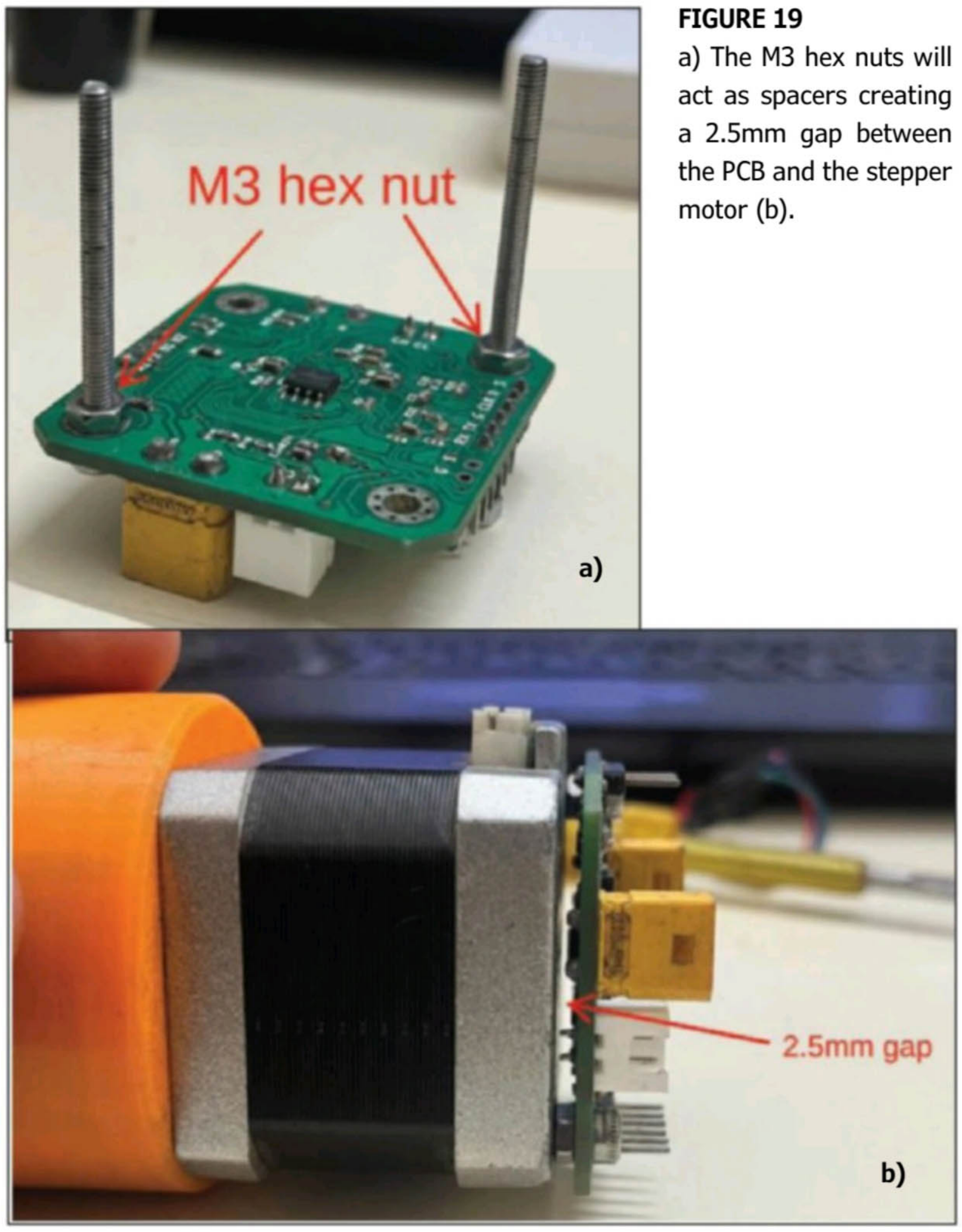


FIGURE 19
a) The M3 hex nuts will act as spacers creating a 2.5mm gap between the PCB and the stepper motor (b).

    https://github.com/stm32duino/BoardManagerFiles/raw/main/package_stmicroelectronics_index.json
    - Click OK.
    - Then go to Tools >> Board >> Boards Manager, search for STM32, and install it.
    - In Arduino IDE, go to Tools >> Board, and select your STM32 series (for example, Generic STM32F1 Series).
    - Under Tools >> Port, select the COM port that corresponds to your STM32 device.

2) Download and install the necessary libraries:
   a) Download the necessary libraries from the following URLs:
      - https://github.com/adafruit/Adafruit_INA219
      - https://github.com/adafruit/Adafruit_MPU6050
      - https://github.com/RobTillaart/AS5600
      - https://github.com/sandeepmistry/arduino-CAN/tree/master/examples
   b) Add the library to Arduino IDE.
      - Open the Arduino IDE.
      - Go to Sketch >> Include Library >> Add .ZIP Library
      - Navigate to the folder where you downloaded the .zip library file.
      - Select the .zip file and click OK.

   The library will now be installed and available in the Arduino IDE.

3) Upload the Code.
   a) Connect the STLINK Programmer to the PCB: connect the SWDIO pin of the programmer to the D pin on the PCB, the SWCLK pin of the programmer to the CLK pin on the PCB, and the ground pin of the programmer to the PCB ground (refer to circuit diagram shown in Figure 14).
   b) Go to TOOLS on the Arduino IDE, and select the upload method as an STM32 cube programmer (SWD).
   c) Download the robotic actuator code with .ino file
   d) Open the code you downloaded and compile.
   e) Click the Upload button (right arrow icon) in the Arduino IDE.
   f) Wait for the IDE to compile, and upload the code to the STM32. You'll see a message that says "Upload Complete" once the process finishes.

FINAL SETUP AND TESTING OF THE ROBOTIC ACTUATOR

We used two connectors for power and two for data, to allow multiple actuators to be connected in series, helping to reduce messy wiring in the robotic system. Both data lines are connected in parallel, as are both power lines. The yellow color XT30 connectors for power supply and the white color 2-pin JST connector for CAN communication (**Figure 20**).

Use the USB-to-CAN converter to communicate with the robotic actuator. Connect the 2-pin JST connector to either J4 or J6 on the actuator module, then connect the CH pin to the CAN HIGH pin on the USB-to-CAN converter, and the CL pin to the CAN LOW pin on the converter.

Connect the USB-to-CAN converter to the laptop, open the serial monitor, and send commands to control the rotational movement of the actuator. You can also retrieve data such as RPM, direction, power consumption, and orientation. The following commands control the actuator and obtain sensor data via the CAN bus:

CLOC—Controls the actuator in the clockwise direction.
ACLOC—Controls the actuator in the anticlockwise direction.
VOLT—Retrieves the input voltage of the actuator
CURRENT –Retrieves the current consumption of the actuator.
DIR—Gets direction of the actuator (clockwise/anticlockwise).

ABOUT THE AUTHORS

Maheshwaran Sadasivam is an electronics and robotics enthusiast who completed his MSc in Electronics from St. Joseph's College Tiruchirappalli, Tamilnadu, India. He is currently working as Lead Prototype Engineer at Waltr, Bangalore, Karnataka, India. His field of interests are human-centric automation systems, additive manufacturing, and robotics.

Vairamani Kanagavel is an electronics researcher who completed his MSc in Electronics from St. Joseph's College, Tiruchirappalli, Tamilnadu, India and PhD in Instrumentation from Madurai Kamaraj University, Madurai, Tamilnadu India. His field of interests are wireless sensor networks and human-centric automation systems.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials

RESOURCES

Adafruit | www.adafruit.com

Microchip Technology | www.microchip.com

STMicroelectronics | www.st.com

Texas Instruments | www.ti.com

RPM—Gets the rpm of the actuator.

ANGLE—Gets the angular position of the output shaft of the actuator.

ORIEN—Gets data from the accelerometer and gyroscope sensors to detect the orientation of the actuator.

Based on these commands, the actuator will interact with the master controller or laptop. As previously mentioned, we can opt to connect multiple actuators in a daisy-chain configuration, with each actuator having a unique CAN ID. This setup allows for the control of all actuators using just the two CAN lines—CAN HIGH and CAN LOW.

We have developed the code for these eight commands. And, developers can modify and integrate the code according to their specific application requirements.

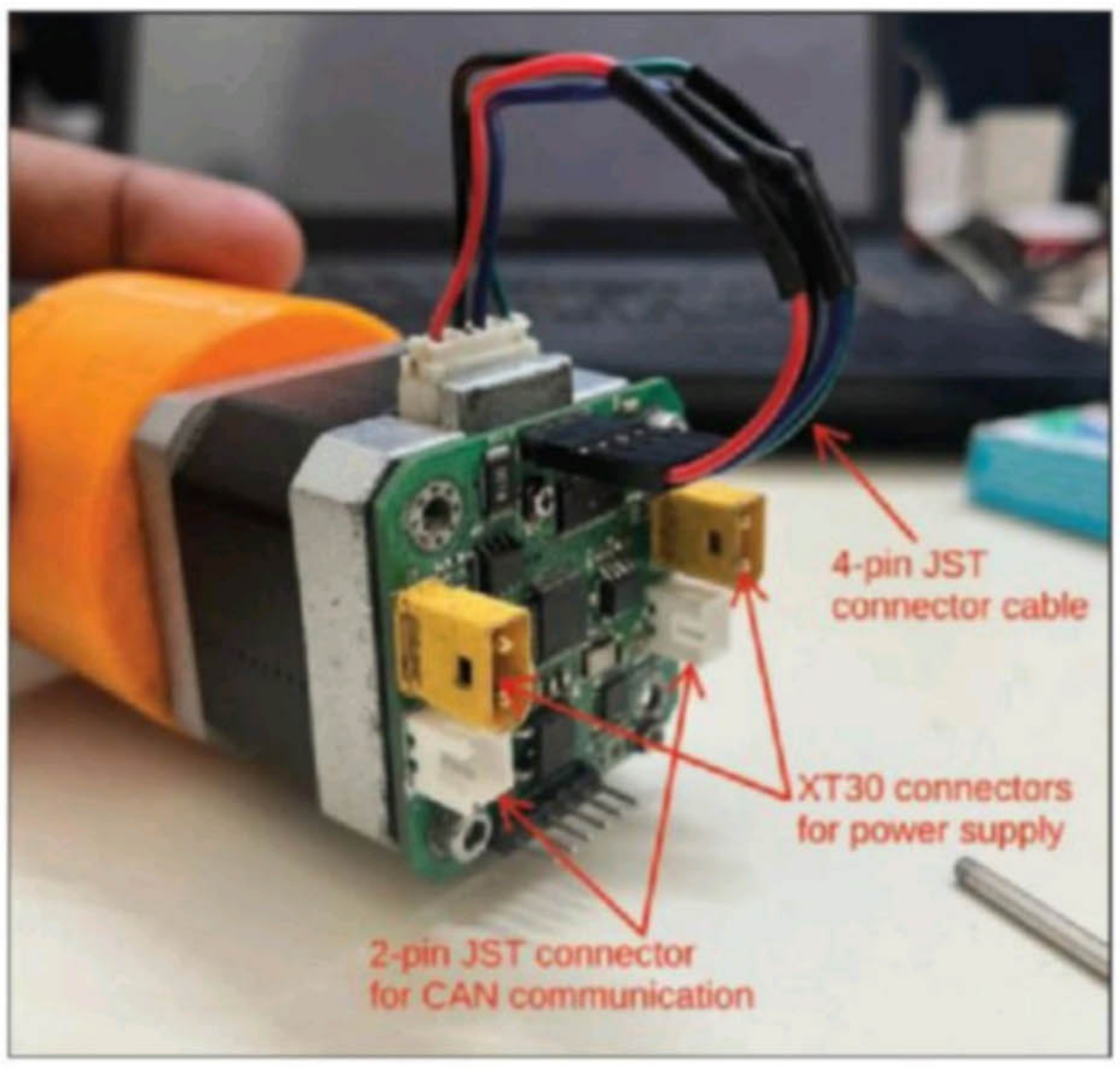
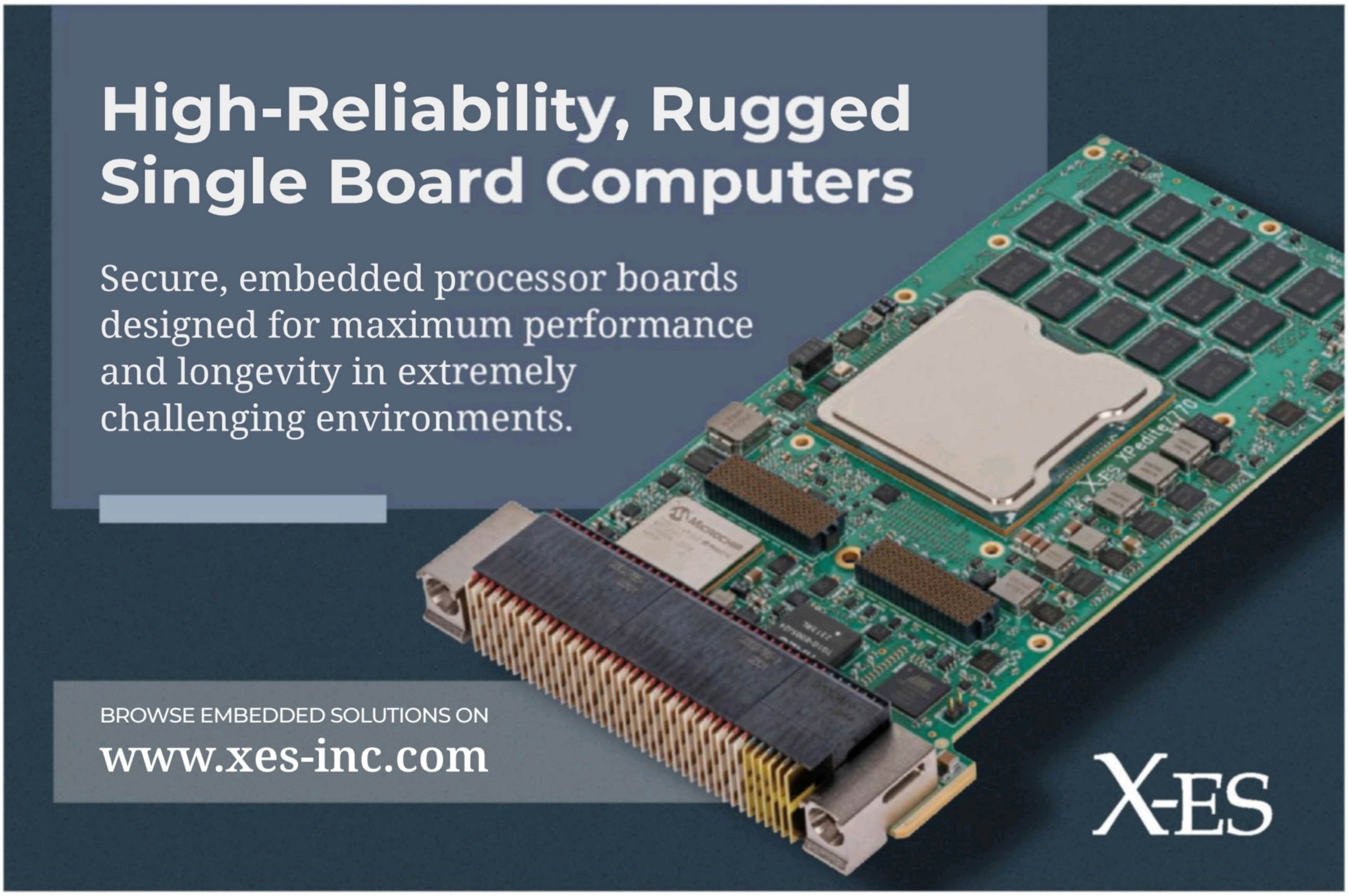

4-pin JST connector cable
XT30 connectors for power supply
2-pin JST connector for CAN communication

FIGURE 20
The yellow color XT30 connectors are used for the power supply and the white color 2-pin JST connectors are used for CAN communication.

CONCLUSIONS

The development of this cost-effective, 3D-printed robotic actuator offers a practical and accessible solution for students and researchers in the field of robotics. By combining a stepper motor, a 3D-printed planetary gearbox, and an array of sensors for real-time feedback, the actuator provides precise motion control and efficient power usage. Its modular design, facilitated by the CAN bus protocol, allows for easy integration into various robotic systems.

Experimental results demonstrate the actuator's high positional accuracy, energy efficiency, and adaptability, proving its potential as a valuable component in advancing both research and real-world robotic technologies. With its scalable, low-cost design, this actuator serves as a foundation for further innovation and experimentation in robotics.


High-Reliability, Rugged Single Board Computers

Secure, embedded processor boards designed for maximum performance and longevity in extremely challenging environments.

BROWSE EMBEDDED SOLUTIONS ON
www.xes-inc.com

X-ES

A 3-Phase Energy Meter with Multicore Support

Using an ESP32 MCU with Voltage and Current Sensors

By
Gamal Labib

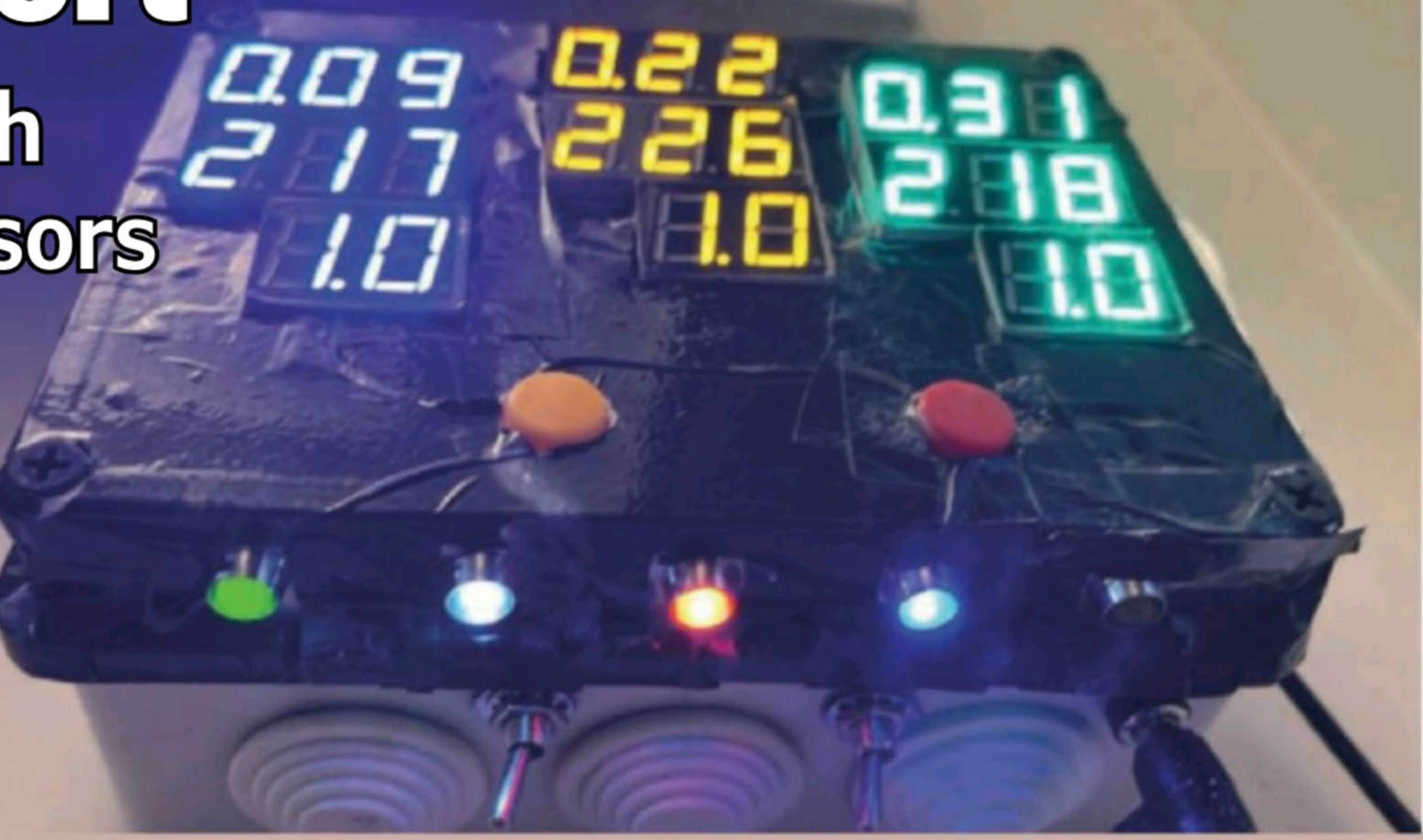


FIGURE 1
The exterior structure of the project. Top: three 7-segment 8-digit modules, each of which is allocated to a specific mains supply phase. Mid-right: an MCU reset red button, a yellow push button to display button. Mid-bottom: five LED indicators (from right)—Blink, Wi-Fi connectivity, 5V power supply, Activity, and External MCU power source (for example, Laptop's USB port). Bottom right: DC 5V jack, MCU power source selector switch, Continuous Display switch.

To continuously monitor energy consumption in his home, and store and communicate the data wirelessly, Gamal built a 3-phase energy meter, using a single ESP32 MCU with dual-core architecture, and voltage and current sensors. In his novel design, one core senses voltage and current and calculates consumed energy, while the other, long-forgotten core handles the user's web requests, logs measurements on an SD card, and displays current, voltage, and cumulative power consumption.

Understanding the energy consumption pattern of home appliances and gadgets is important for owners to optimize and to minimize their electricity bills. Keeping track of the official energy meter to identify our daily consumption may prove impractical, especially if the meter is installed outside the premises. I needed an inexpensive solution to adjust my electricity consumption and to estimate future bills.

I came across many energy meter projects based on microcontrollers (MCUs) and sensor modules, but they mostly focused on single-phase mains power supplies. Since I have 3-phase mains, I would then require three homemade meters and offline analysis of their readings to manage my consumption. In this project, I consolidate the mains supply monitoring for my home into a single MCU-based meter for which I can embed all analysis code once, and control it within my home Internet of Things.

I chose the ESP32 MCU for this project for several reasons. It supports Wi-Fi and Bluetooth for remote access. It supports many analog GPIOs required by the voltage and current sensors. And above all, it has dual core architecture, a feature rarely found in its price range. It looks like embedded systems designers overlooked the duality-of-core feature and kept using 50% of the MCU processing capabilities, at least for energy meter projects.

I considered the duality of cores a prerequisite for a 3-phase meter, since processing voltage and current data is triple that of the regular workload of a single-phase meter. Furthermore, serving user's queries and commands would interrupt the energy measurement process, so it requires a separate core to guarantee consistency. The ESP32 provides an abundance of memory that enables it to run multimedia and AI applications. This makes it a good fit for this project which is rather large in code size. It takes around 80% of the ESP32's 1.25MB program memory, indicated in the Arduino IDE compilation output for the project's sketch as indicated:

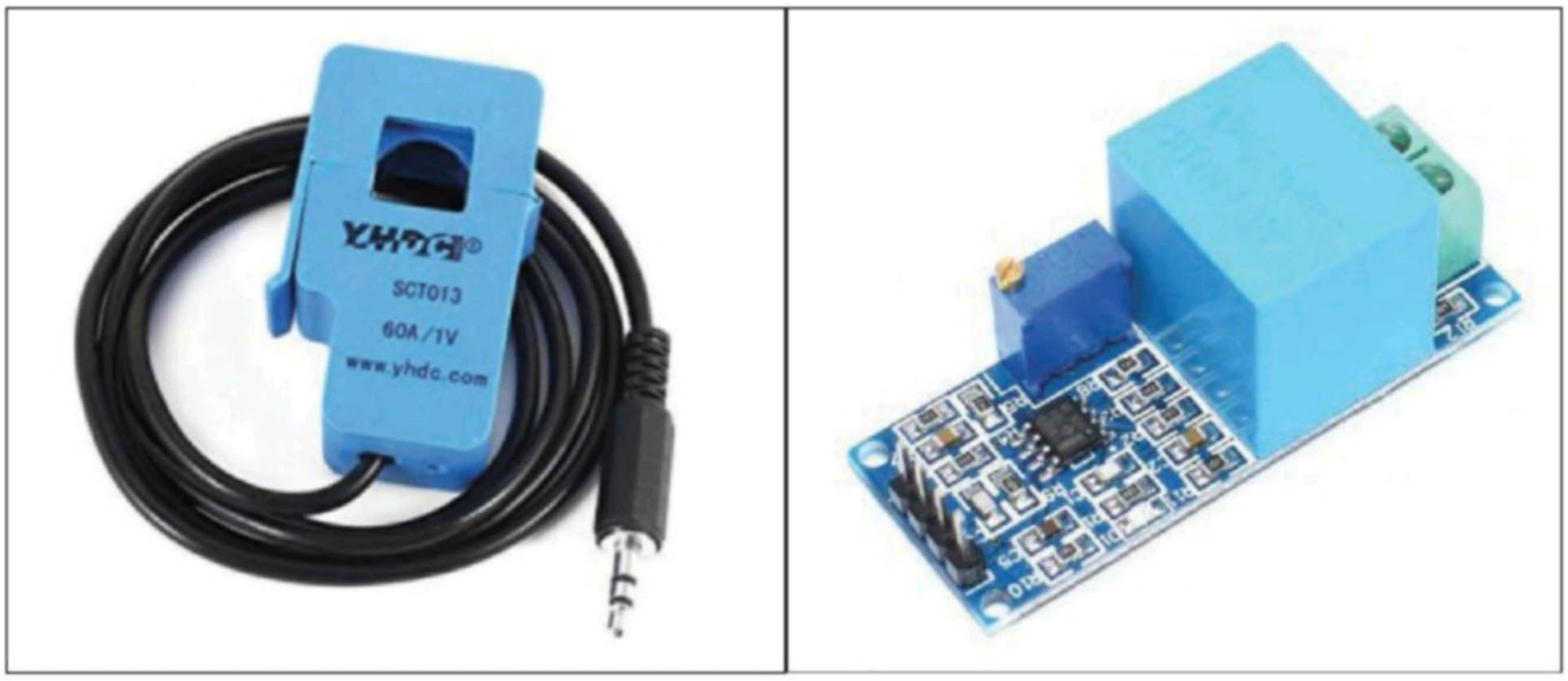


FIGURE 2
The ZMPT101B voltage sensor module (left) and SCT-013 60A/1V current sensor transformer (right).

Sketch uses 1,063,197 bytes (81%) of program storage space. Maximum is 1,310,720 bytes.

Global variables use 50,516 bytes (15%) of dynamic memory, leaving 277164 bytes for local variables. Maximum is 327,680 bytes.

HARDWARE

The meter project is housed in a waterproof plastic box 15cm×11cm×8cm and is built around the ESP32-WROOM-32U development board, alongside three ZMPT101B [1] voltage sensing modules [2], an SD Card module, and three 7-segment 8-digit display blocks [3] (**Figure 1**). There are also three SCT-013 60A/1V AC current sensors [4] external to the meter and hooked to the mains power supply feeding cables (**Figure 2**). A sample display per phase for the cumulative consumed power (in one-hundredth accuracy), the instant AC voltage and current (in one-tenth accuracy) is shown in **Figure 3**.

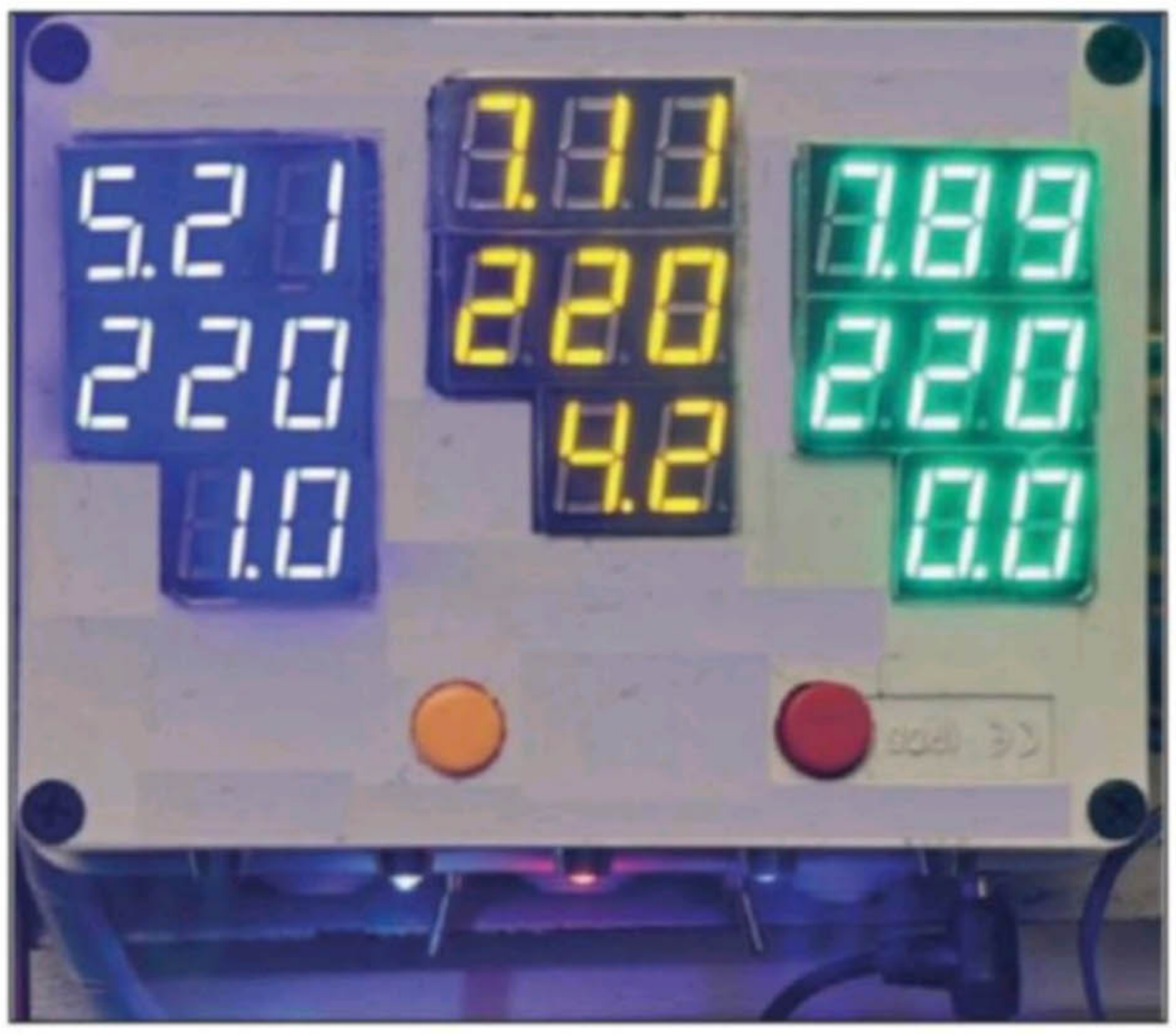


FIGURE 3
The three 7-segment display blocks—one block per mains power phase. The top line shows the consumed energy in kilowatt hours (kWh), the middle is for voltage (V), and the bottom shows the current in Amperes (A).

Figure 4 depicts the peripherals wiring to the MCU. Their VCC is fed directly from the external 5V DC adapter, except for the current sensing bridge, which is powered from the MCU's onboard 3.3V regulator. **Figure 5** illustrates the wiring of the project sensors to my home electrical switching panel, which became a little bit cluttered following some modifications to balance the load distribution on the 3-phase supply. I used a clamp-type current meter to help me tune and verify the driving sketch code of my current sensors.

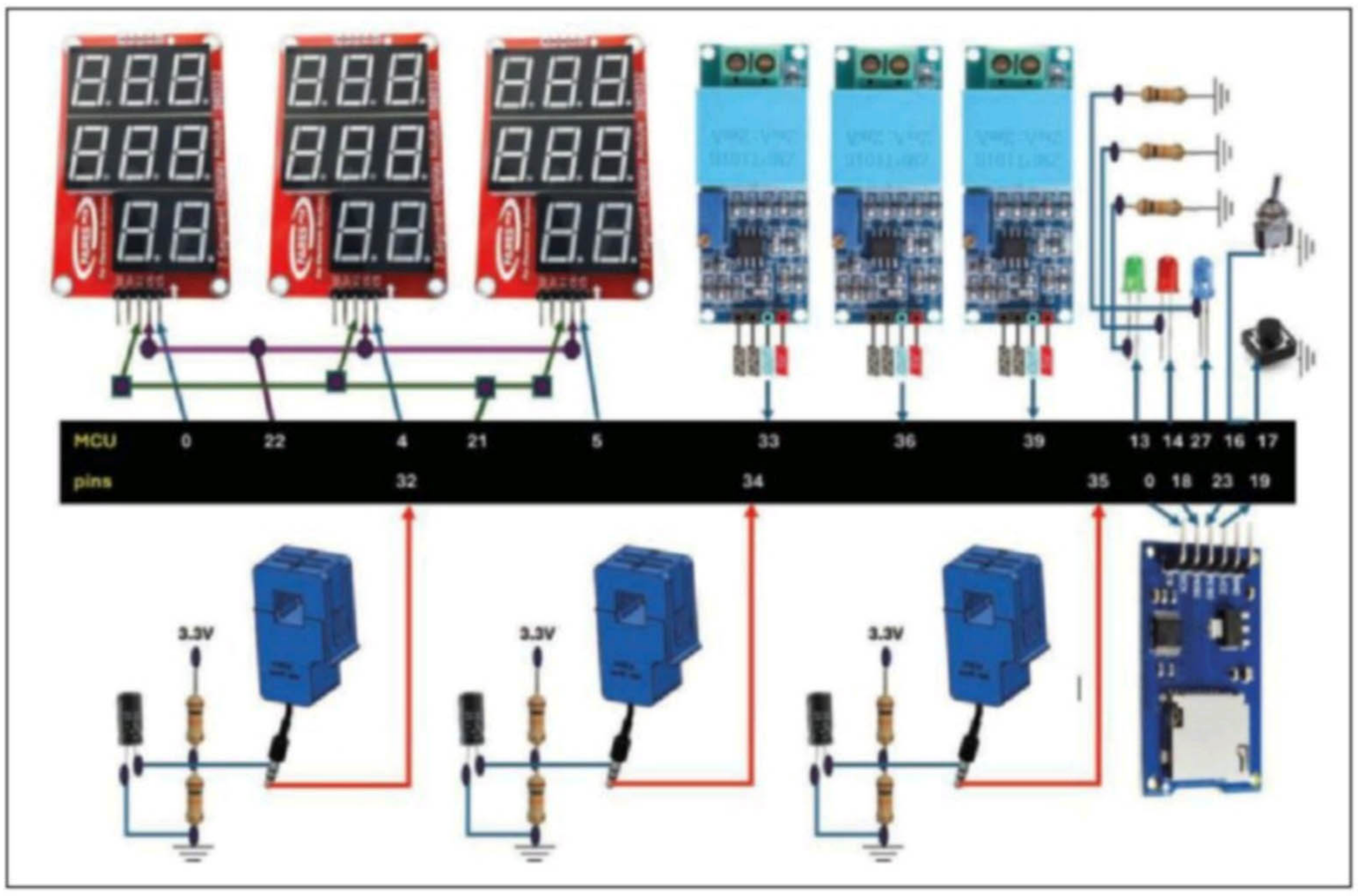


FIGURE 4
Wiring the display and voltage and current sensing modules to the MCU. The 3.3V feed to the current sensing bridge (10kΩ, 10µF) is fed from the MCU development board. The LEDs are wifiSTAT (green), ACTIVITY (red), and BLINK (blue), and they use 220Ω resistors. The selector switch forces continuous display (`segON` in code), and the push button enables on-demand display for a specified duration (`segSW` in code).

WORKING WITH THE ESP32'S DUAL CORE ARCHITECTURE

This MCU has two identical cores—Core-0 and Core-1—sharing its program memory and RAM. They both have equal access to the device's GPIOs and thus to its connected peripherals. The default core used in single core applications is Core-1. To utilize both cores, we need to code the device using the real-time operating system (RTOS) supplied by the MCU manufacturer, and that one is

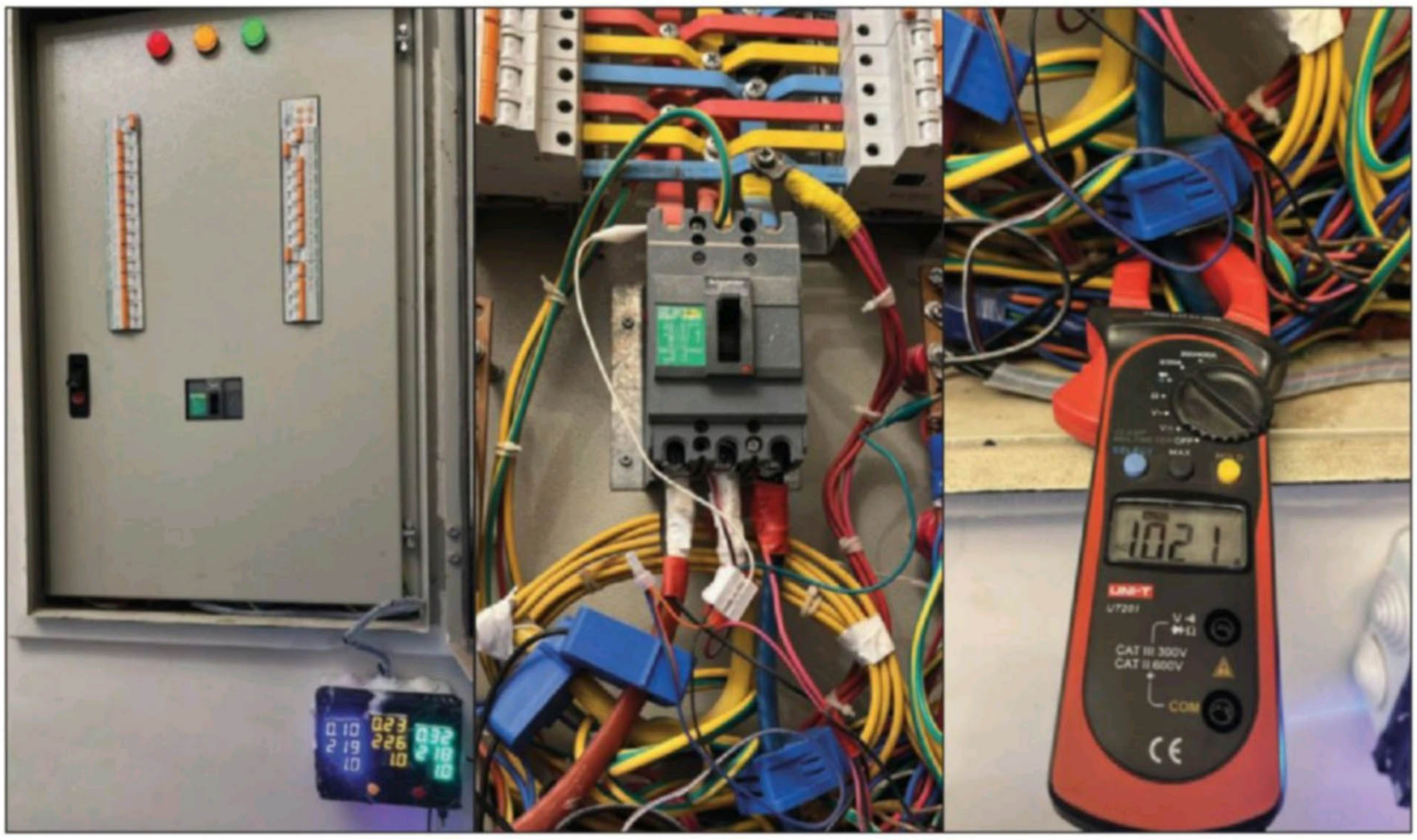


FIGURE 5
Wiring the project box to the mains power supply fuse panel (left). Wiring the blue current sensors (middle). Verifying current measurement using AC current multimeter (right).

already built into the ESP32 hardware core for the Arduino IDE v.2.3.2. So, instead of building a traditional sketch composed of a `setup()` and a `loop()` section, we need to forget about putting any code in the latter section, and just declare two special functions (called "***tasks***" in RTOS terminology) in the `setup()` section. Each task configures the resources allocated to a specific core. The implementation of those functions in the sketch contains all the code we need to set up the hardware under the control of the relevant core at MCU startup, and also the code that will continue to run by the core throughout the uptime of the MCU.

I always maintain modularity when writing complex project code, since it helps debugging and reusability. This explains why the structure of the project sketch is composed of six files, alongside the main ***ino*** file, in which we declare the two tasks that will run on the MCU cores. Each file contains declarations and functions relevant to a specific peripheral. I identified the peripherals as the SD card module (***SD_Functions.h***), the MCU built-in Wi-Fi module (***Wlan.h***), the display modules (***Display.h***), and the clock module (***Clock.h***), in addition to the two cores (***task0.ino*** and ***task1.ino***).

Referring to **Listing 1**, we see the invocation of RTOS by declaring the two tasks of class type `TaskHandle_t`, then in the `setup()` section we emphasize their configuration in `xTaskCreatePinnedToCore()` functions. A task configuration includes the name of the relevant task implementation (for example, `Task0code` and `Task1code`), which will be detailed in separate ***ino*** sketch files, the task name (for example, `Task0` and `Task1`), the task's core ID (for example, `0` and `1`), the task's priority on its assigned core, and the task's stack size. I did some trial and error work for setting up the latter two configuration settings. The highest priority is set to 1, while increasing its value may cause some activities to break—such as wireless LAN setup, which fails at priority 100. As for the stack size, I found that 100,000 bytes is the achievable maximum beyond which the core fails to work, whereas 10,000 bytes is an operable setting for this project.

TASK#0 RUNNING ON CORE-0

Looking into the implementation of each task, refer to **Listing 2**, which illustrates the setup section of the task#0 for handling the SD card module, the wireless LAN, the Web application server, the NTP service, and the 7-segment display. The SD card module is hooked to VS-SPI of the MCU and is used to keep a log file of periodically sensed measurements (for example, every 15 minutes), and a summary file for the accumulated power consumption per mains supply phase. Wi-Fi is required to link the meter to the Internet for adjusting local timestamps of logged data by synching with a global NTP server, and for maintaining a Web server to serve the user's requests.

ABOUT THE AUTHOR
Gamal Labib has been an enthusiast of embedded systems for two decades and currently a mentor (at codementor.io). He holds a MEng and a PhD in IT. Besides writing for technical magazines, he is a visiting associate professor at Egyptian universities, and a certified IT consultant. Email: drgamallabib@yahoo.co.uk

The server handles a set of client requests, such as listing logged data, listing accumulated power consumption per phase, resetting its calculation to zero, and erasing log file (that is, recycling the file). **Figure 6** illustrates some Web client queries to the energy meter, using its IP address (in my case it is “192.168.1.182”), or its DNS host name (in my case it is “energy-3p.local”). The Web server responds to a few keywords in the client’s URL queries such as “***sum***” to show the accumulated power consumption in ***kWh*** per phase from the summary file, “***list***” to display the log file records, “***clear***” to erase the summary file, and “***erase***” to erase the log file.

Listing 3 shows the loop section of Task#0 marked by the for(;;) statement, by which Core-0 continuously checks for Web client requests to serve, and periodically logs sensed data into the SD card, and displays sensed data on the 7-segment display.

The benefit of dedicating a MCU core to run those activities can be demonstrated simply when the user requests a list of thousands of logged data records from the SD card without disrupting the current and voltage sensing activity, thus power consumption calculations become credible.

Activating the 7-segment display is designed to work in two modes: continuously showing logged data by turning on the designated selector switch (shown in Figure 1 and Figure 4), which is wired to GPIO16 (for example, the `segON` in **Listing 3**) of the MCU, or manually by pressing the yellow push-button (Figure 1), which is wired to GPIO17 (for example, `segSW`). I coded the Blink and the Activity LEDs to fluctuate interchangeably to give the feeling of system alive, while fluctuation is suspended during the serving of Web client requests and resumes afterward.

Note that working with multicore architecture required re-initializing the 7-segment display blocks prior to showing any data for each round of the loop section of the task.

TASK#1 RUNNING ON CORE-1

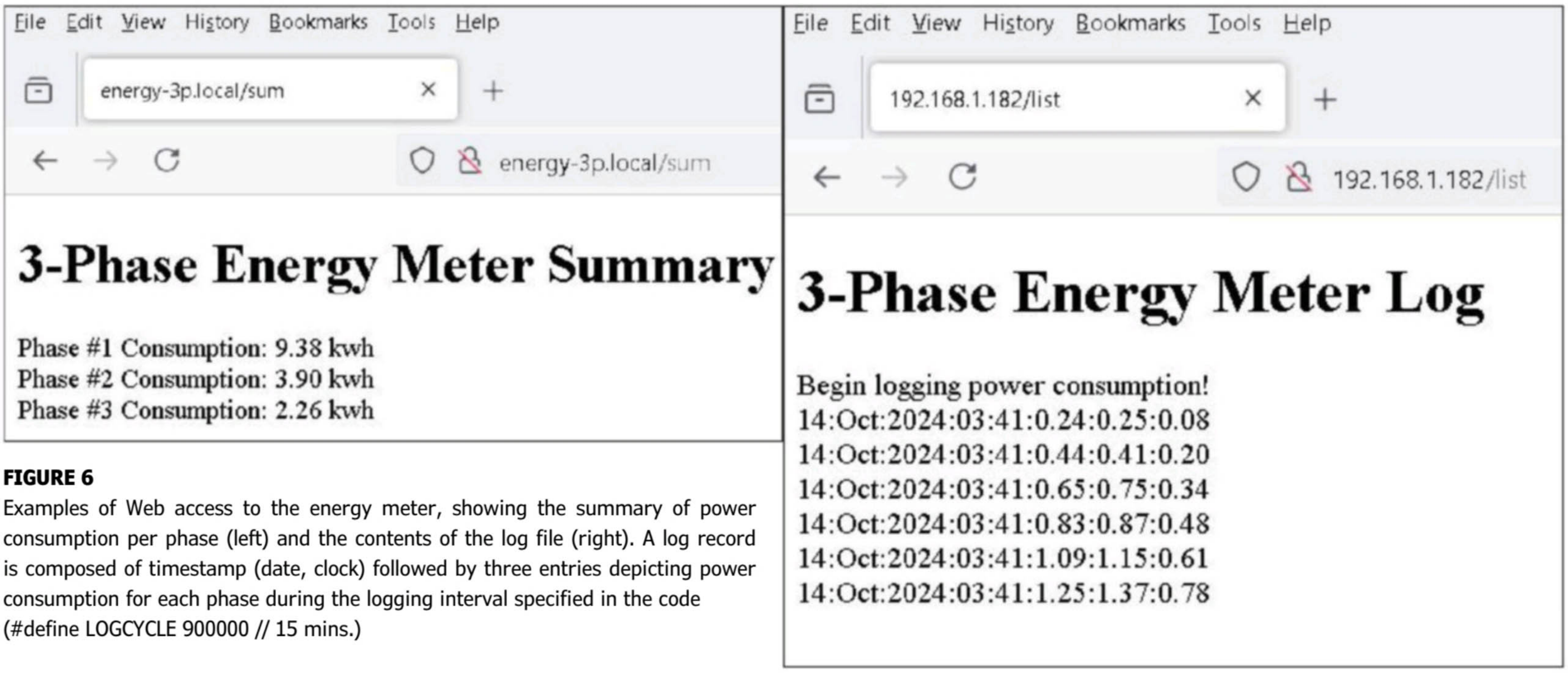
Task#1 is responsible for continuously sensing AC current and voltage, and for calculating the power consumption. Monitoring the AC current using the SCT-013 sensor is based on the ***Emonlib*** library [5]. I emulated the Simply Explained blog [6] that dealt with a single phase mains supply, and I modified their code to fit my objective of handling a 3-phase source (see **Listing 4**). I faced some difficulty adjusting the resistor bridge to get AC current readings matching those of the clamped multimeter. I settled on a resistor value of 10kΩ for the voltage divider circuit, but still got some current reading of few hundred milliamps at no load condition.

```
TaskHandle_t Task0;
TaskHandle_t Task1;

void setup() {
  analogReadResolution(10);

  pinMode(segSW, INPUT_PULLUP);  // toggle switch 7-segments
display ON/OFF
  pinMode(segON, INPUT_PULLUP);  // selector switch
7-segments display ON

  pinMode(wifiSTAT, OUTPUT);  // state of wiFi connection
  pinMode(ACTIVITY, OUTPUT);  // ACTIVITY in sensing
  pinMode(BLINK, OUTPUT);     // blink periodically
  delay(200);

  digitalWrite(wifiSTAT, LOW);
  digitalWrite(ACTIVITY, LOW);
  digitalWrite(BLINK, HIGH);

  Serial.begin(115200);

  xTaskCreatePinnedToCore(
    Task0code, /* Task function. */
    “Task0”,   /* name of task. */
    10000,    /* Stack size of task */
    NULL,      /* parameter of the task */
    1,         /* priority of the task */
    &Task0,    /* Task handle to keep track of created task
*/
    0);        /* pin task to core 0 */
  delay(500);

  xTaskCreatePinnedToCore(
    Task1code, /* Task function. */
    “Task1”,   /* name of task. */
    10000,     /* Stack size of task */
    NULL,      /* parameter of the task */
    1,        /* priority of the task */
    // when priority was set to 100 then network scan would
not complete
    &Task1, /* Task handle to keep track of created task */
    1);     /* pin task to core 1 */
  delay(500);
}
```

LISTING 1
The `setup()` section of the sketch declaring the shared GPIOs between cores and the RTOS task assigned to each core. The implementation of those tasks is detailed in two separate header files within the sketch.

```
void Task0code(void* pvParameters) {
  Serial.print(“+++++++++++++++++++++ Task0 running on core “);
  Serial.println(xPortGetCoreID());

  sd_setup();
  checkFiles();
  wlan_setup();
  web_setup();
  show_time();
  initialise();  // initialize 7-segments displays
  delay(2000);
```

LISTING 2
The setup section of Task#0 for the SD card, the wireless LAN, the 7-segment display, and the Web application.


File Edit View History Bookmarks Tools Help
energy-3p.local/sum
energy-3p.local/sum
File Edit View History Bookmarks Tools Help
192.168.1.182/list
192.168.1.182/list

3-Phase Energy Meter Summary

Phase #1 Consumption: 9.38 kwh
Phase #2 Consumption: 3.90 kwh
Phase #3 Consumption: 2.26 kwh

3-Phase Energy Meter Log

```
Begin logging power consumption!
14:Oct:2024:03:41:0.24:0.25:0.08
14:Oct:2024:03:41:0.44:0.41:0.20
14:Oct:2024:03:41:0.65:0.75:0.34
14:Oct:2024:03:41:0.83:0.87:0.48
14:Oct:2024:03:41:1.09:1.15:0.61
14:Oct:2024:03:41:1.25:1.37:0.78
```

FIGURE 6
Examples of Web access to the energy meter, showing the summary of power consumption per phase (left) and the contents of the log file (right). A log record is composed of timestamp (date, clock) followed by three entries depicting power consumption for each phase during the logging interval specified in the code (#define LOGCYCLE 900000 // 15 mins.)

I worked around this by ignoring readings below 500mA and rounding whatever was above that to 1.0A. I put this statement in the code to do that:

```
if (Irms < 1.0) Irms = round(Irms);
```

I assume such an approximation does not affect the legitimacy of my energy meter project, since my concern is tracking heavy loads that bring bulky bills. Furthermore, I discovered some facts about my appliances' consumption of energy, and this led me to pay attention to economizing their usage. For example, the microwave and 1hp water pump are rated at [5.0A ≈ 1hp Air Conditioner], whereas the solar water heater (in winter) and an electric kettle require 10.0A.

As for AC voltage measurement using the ZMPT101B voltage sensor module, the calculations are based on the Filters library [7]. I followed the example of the SURTR TECHNOLOGY blog [8], but still also tailored their code to fit the 3-phase power source. Coarse calibration of the voltage module can be done using its built-in potentiometer, and this is necessary for each phase; then fine tuning the readings should be done by adjusting the value in this statement in Listing 4:

```
current_Volts1 = current_Volts1 * (48);
```

```
    for (;;) {
      server.handleClient();

      if (digitalRead(segSW) == 0 && digitalRead(segON) == 1) {
        initialise();
        show_Data();
      }

      ++show_timer0;
      if (show_timer0 >= SHOWCNT) {
        digitalWrite(ACTIVITY, !digitalRead(ACTIVITY));
        digitalWrite(BLINK, !digitalRead(BLINK));

        show_timer0 = 0;
        if (digitalRead(segON) == 0) {
          initialise();
          show_Data();
        }
        if (digitalRead(segSW) == 1 && digitalRead(segON) == 1) initialise();
      }

      process_logfile();  // invokes processing log and summary files
      // ==================================================
    }
  }
```

LISTING 3
The loop section of Task#0 for continuous handling of the 7-segment displays and the sensed data files stored on an SD card, and user access via the Web application.

```
void Task1code(void* pvParameters) {
  Serial.print(“====================== Task1 running on core “);
  Serial.println(xPortGetCoreID());

  analogReadResolution(10);

  emon1.current(iPORT1, 60);  // Current: input pin, calibration
  emon2.current(iPORT2, 60);  // Current: input pin, calibration
  emon3.current(iPORT3, 60);  // Current: input pin, calibration

  RunningStatistics inputStats1;
  RunningStatistics inputStats2;
  RunningStatistics inputStats3;
  inputStats1.setWindowSecs(windowLength);
  inputStats2.setWindowSecs(windowLength);
  inputStats3.setWindowSecs(windowLength);

  for (;;) {
    Serial.print(“====================== Task1 running on core “);
    Serial.println(xPortGetCoreID());

    while (true) {
      Sensor = analogRead(vPORT1);  // read the analog in value:
      inputStats1.input(Sensor);    // log to Stats function
      Sensor = analogRead(vPORT2);  // read the analog in value:
      inputStats2.input(Sensor);    // log to Stats function
      Sensor = analogRead(vPORT3);  // read the analog in value:
      inputStats3.input(Sensor);    // log to Stats function

      if ((unsigned long)(millis() - previousMillis) >= printPeriod) {
        Serial.print(“====================== Task1 running on core “);
        Serial.println(xPortGetCoreID());
        previousMillis = millis();  // update time every second

        current_Volts1 = intercept + slope * inputStats1.sigma();  //Calibartions for offset and 
amplitude
        current_Volts1 = current_Volts1 * (48);                    //Further calibrations for the 
amplitude
        current_Volts2 = intercept + slope * inputStats2.sigma();  //Calibartions for offset and 
amplitude
        current_Volts2 = current_Volts2 * (48);                    //Further calibrations for the 
amplitude
        current_Volts3 = intercept + slope * inputStats3.sigma();  //Calibartions for offset and 
amplitude
        current_Volts3 = current_Volts3 * (48);                    //Further calibrations for the 
amplitude

        Irms1 = emon1.calcIrms(1480);           // Calculate Irms only
        if (Irms1 < 1.0) Irms1 = round(Irms1);  // ignore small loads

        RMSPower1 = current_Volts1 * Irms1;  //Calculates RMS
        if (RMSPower1 > peakPower1) peakPower1 = RMSPower1;
        kilos1 = kilos1 + (RMSPower1 * (2.05 / 60 / 60 / 1000));  //Calculate kilowatt hours used
```

LISTING 4
Task#1 runs on Core-1 for sensing voltage and current of each mains supply phase, and for calculating the corresponding consumed energy per second. The hourly consumed power is then calculated by scaling the per-second calculation to per-hour. Irms, RMSPower, and kilos for phases 2 and 3 are replicates of the shown phase 1 calculations.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] to [8] as marked in the article can be found there.

RESOURCES

Fares PCB | www.fares-pcb.com

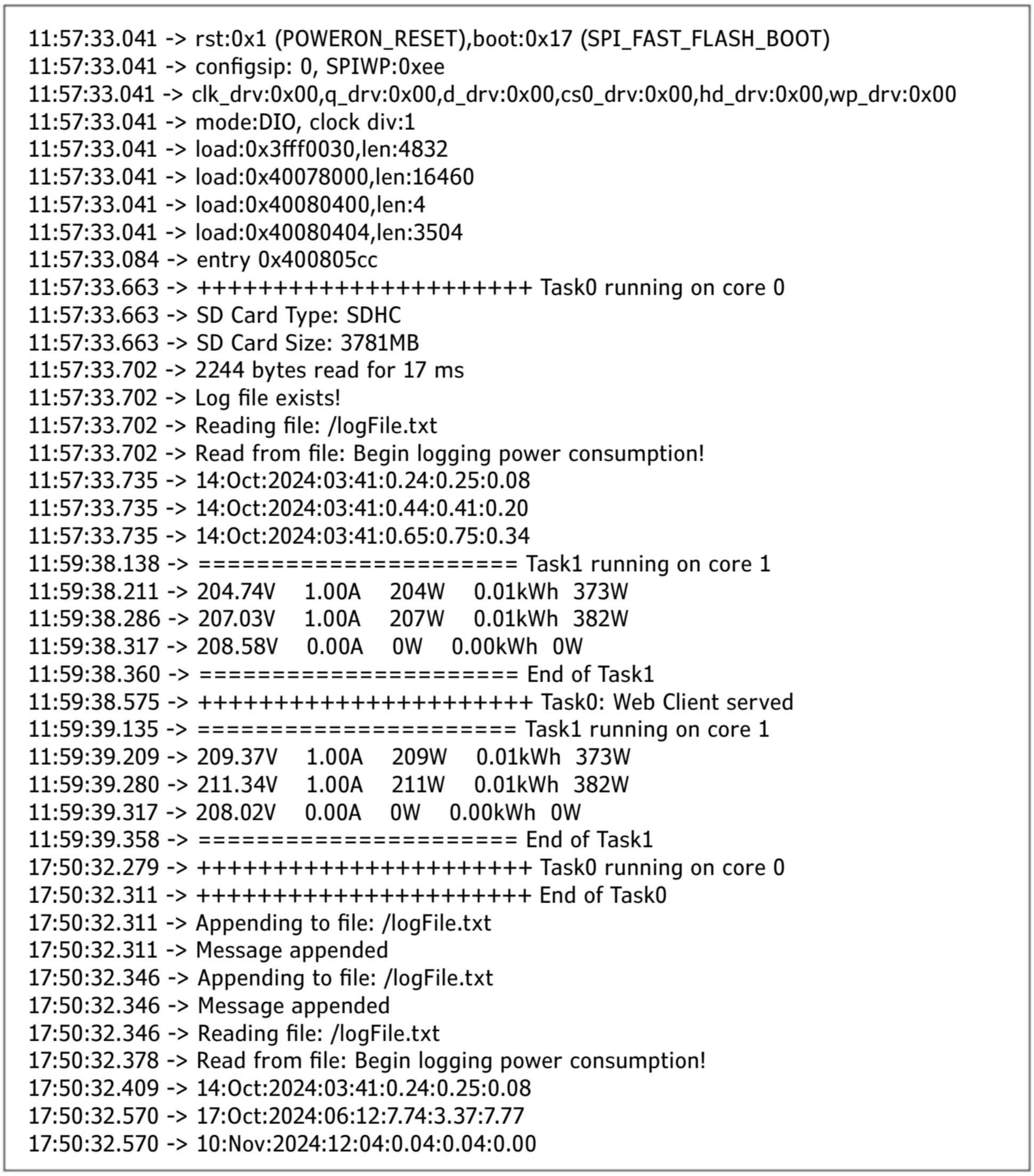
You might notice fluctuations in the voltage readings of each phase, even though they are almost identical. I assume this case is somehow an exaggeration of what you see in digital multimeters that show minor reading fluctuations. Furthermore, I came across some energy meter projects in which the mains voltage is fixed to some value (such as 230V) to simplify the design.

A printout on the Arduino IDE serial monitor for the meter in action is shown in **Figure 7**, in which the MCU startup and run for Core-0 and Core-1 activities are shown. Task#0 shows its preliminary check of SD Card files, then at some point, it confirms serving a Web client request, and later on it appends a log record to the log file. Task#1 shows its acquired measurements for instant voltage, current, energy, kWh per-second, and maximum recorded energy.

This project draws a high current of more than 4A due to the nature of the ESP32 itself, and for the extensive peripherals I put into it, including the 7-segment displays, the SD card module, and the voltage-sensing modules. Smaller DC current source will disable the operation of the SD Card module and dim the 7-segment display.

CLOSING REMARKS

A 3-phase energy meter design is presented in a novel way, utilizing the dual-core architecture of the ESP32 MCU. One core is dedicated to continuous sensing of AC current and voltage. The other core handles the user's web requests, and manages logging measurements on an SD card and displaying instant AC current and voltage on the 7-segment display alongside cumulative power consumption.



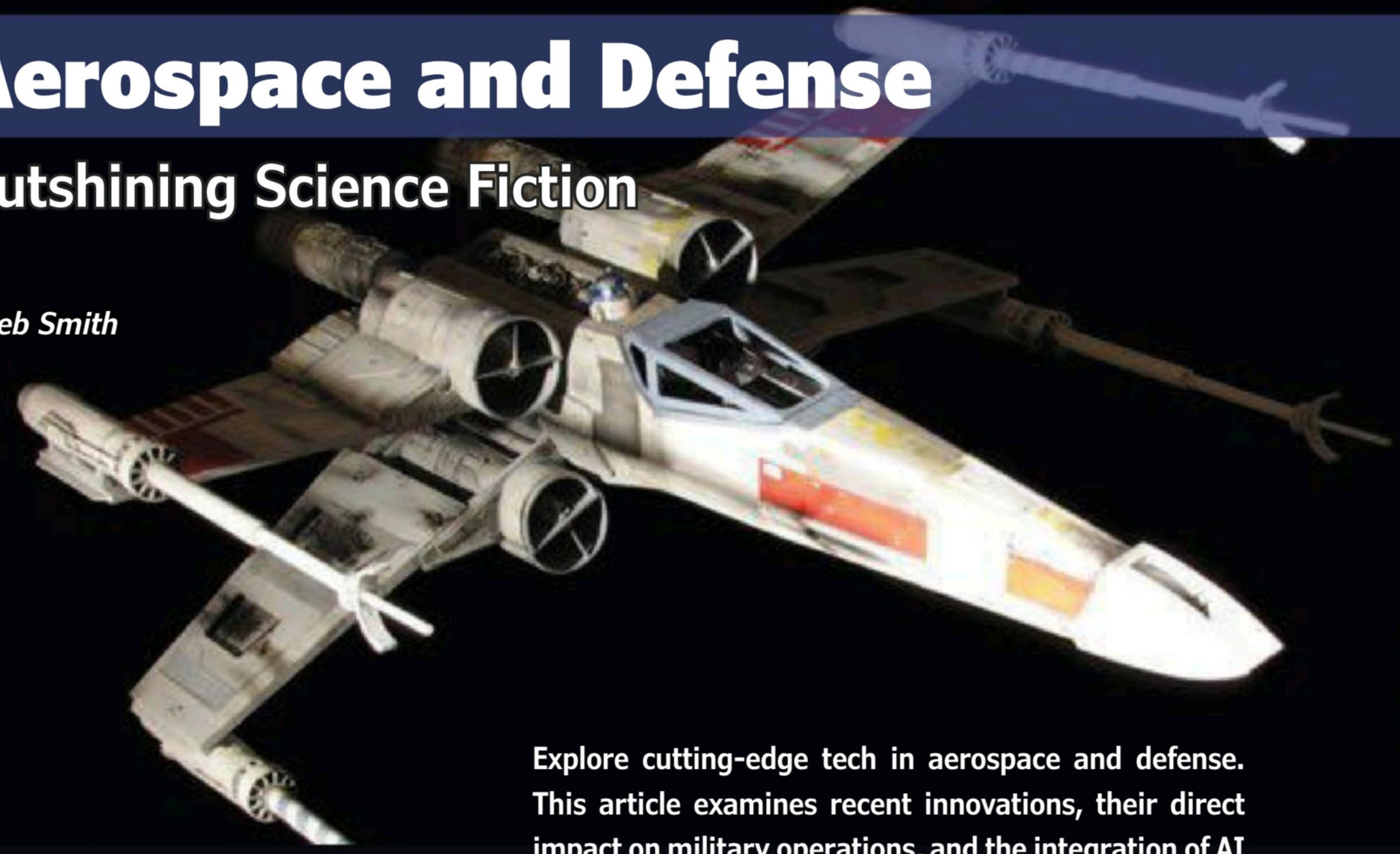
```
11:57:33.041 -> rst:0x1 (POWERON_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
11:57:33.041 -> configsip: 0, SPIWP:0xee
11:57:33.041 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
11:57:33.041 -> mode:DIO, clock div:1
11:57:33.041 -> load:0x3fff0030,len:4832
11:57:33.041 -> load:0x40078000,len:16460
11:57:33.041 -> load:0x40080400,len:4
11:57:33.041 -> load:0x40080404,len:3504
11:57:33.084 -> entry 0x400805cc
11:57:33.663 -> ++++++++++++++++++++++ Task0 running on core 0
11:57:33.663 -> SD Card Type: SDHC
11:57:33.663 -> SD Card Size: 3781MB
11:57:33.702 -> 2244 bytes read for 17 ms
11:57:33.702 -> Log file exists!
11:57:33.702 -> Reading file: /logFile.txt
11:57:33.702 -> Read from file: Begin logging power consumption!
11:57:33.735 -> 14:Oct:2024:03:41:0.24:0.25:0.08
11:57:33.735 -> 14:Oct:2024:03:41:0.44:0.41:0.20
11:57:33.735 -> 14:Oct:2024:03:41:0.65:0.75:0.34
11:59:38.138 -> ====================== Task1 running on core 1
11:59:38.211 -> 204.74V    1.00A    204W    0.01kWh  373W
11:59:38.286 -> 207.03V    1.00A    207W    0.01kWh  382W
11:59:38.317 -> 208.58V    0.00A    0W    0.00kWh  0W
11:59:38.360 -> ====================== End of Task1
11:59:38.575 -> ++++++++++++++++++++++ Task0: Web Client served
11:59:39.135 -> ====================== Task1 running on core 1
11:59:39.209 -> 209.37V    1.00A    209W    0.01kWh  373W
11:59:39.280 -> 211.34V    1.00A    211W    0.01kWh  382W
11:59:39.317 -> 208.02V    0.00A    0W    0.00kWh  0W
11:59:39.358 -> ====================== End of Task1
17:50:32.279 -> ++++++++++++++++++++++ Task0 running on core 0
17:50:32.311 -> ++++++++++++++++++++++ End of Task0
17:50:32.311 -> Appending to file: /logFile.txt
17:50:32.311 -> Message appended
17:50:32.346 -> Appending to file: /logFile.txt
17:50:32.346 -> Message appended
17:50:32.346 -> Reading file: /logFile.txt
17:50:32.378 -> Read from file: Begin logging power consumption!
17:50:32.409 -> 14:Oct:2024:03:41:0.24:0.25:0.08
17:50:32.570 -> 17:Oct:2024:06:12:7.74:3.37:7.77
17:50:32.570 -> 10:Nov:2024:12:04:0.04:0.04:0.00
```

FIGURE 7
Printout on the Arduino IDE serial monitor for the meter in action, showing MCU startup and run, with Core-0 and Core-1 activities. Task#0 shows its preliminary check of SD card files, then at some point, it confirms serving a Web client request, and later it appends a log record to the log file. Task#1 shows its acquired measurements for instant voltage, current, energy, kwh per-second, and maximum recorded energy. (Pay no attention to timestamp flow; I just pasted display segments gathered throughout the day.)

Aerospace and Defense

Outshining Science Fiction

By
Caleb Smith



Explore cutting-edge tech in aerospace and defense. This article examines recent innovations, their direct impact on military operations, and the integration of AI and cybersecurity. Gain insights into future trends that are reshaping the industry and enhancing capabilities.

FIGURE 1
X-Wing fighter, used to blow up the Death Star, a fictional space station and super weapon in the *Star Wars* franchise

In 1977, the first *Star Wars* film was released to theaters, and became one of the most celebrated films of the 20th century, spawning numerous sequels, countless articles of merchandising, books, TV shows, and more.

Between the sci-fi fantasy talks of "the Force," and political machinations of a fictitious galaxy far, far away, the movie depicted an array of dazzling technology brought to life by George Lucas, and the creative and immensely talented workers of Industrial Light and Magic. With ships flying long journeys through space faster than light, the nail-biting finale of the movie hinges upon the apparently near impossibility of hitting a 2m wide exhaust port with computer-guided torpedoes fired from an X-wing fighter (**Figure 1**). It seems fairly disproportionate when compared to jumping to hyperspace or building planet-destroying space stations the size of a small moon. Only about the size of a womp rat, even the creative mind of George Lucas, limited by 1970s technology for his inspiration, concluded that such a shot was only possible via the supernatural guidance of "the Force."

Not even 50 years later though, such a shot, while potentially difficult, could be reliably made with an AGM-114 hellfire missile, reliable to within one meter of its target. This led me to think about the other instances of technology being seemingly outdated in this fantasy space epic. For instance: The initial conflict of the movie arises as Princess Leia, the leader of the Rebel Alliance, must send two droids to hand deliver an approximately 10-second video message and plans for the Death Star. While sending data physically via mailed hard drives was very common in the latter half of the 20th century, and is still a necessity for large data transfers, surely this message and accompanying plans do not represent such a large dataset. These technological shortcomings, along with others presented through this beloved film, lead to only one conclusion: In many important—if not glamorous—ways, we have surpassed the technology presented in *Star Wars*. In this article, I shall demonstrate this fact with explanations of modern aerospace and defense technology, and comparisons to what is shown in the movie in regards to the capabilities of Imperial and Rebel technology.

Since the Greek myth of Icarus and Daedalus, engineers have been fascinated by the concept of human flight. With the launch of the Wright Flyer in Kitty Hawk, NC in 1903 by the Wright Brothers, humans had achieved this dream of powered flight. By World War II, Norden Mk. XV bomb sites (**Figure 2**), small mechanical computers, were being added to bombers to account for speed, direction, and other flight conditions when targeting a drop site with a then astonishing 46 meter circular error rate, while simultaneously acting as an autopilot for the plane. These bombsites were even used by bombardier Colonel Thomas Ferebee in the Enola Gay for the dropping of the Little Boy atomic bomb. Since then, the assistance offered to pilots and planes has grown smaller, become embedded onto chips, and become many thousands of times more powerful, growing into a $136 billion industry.

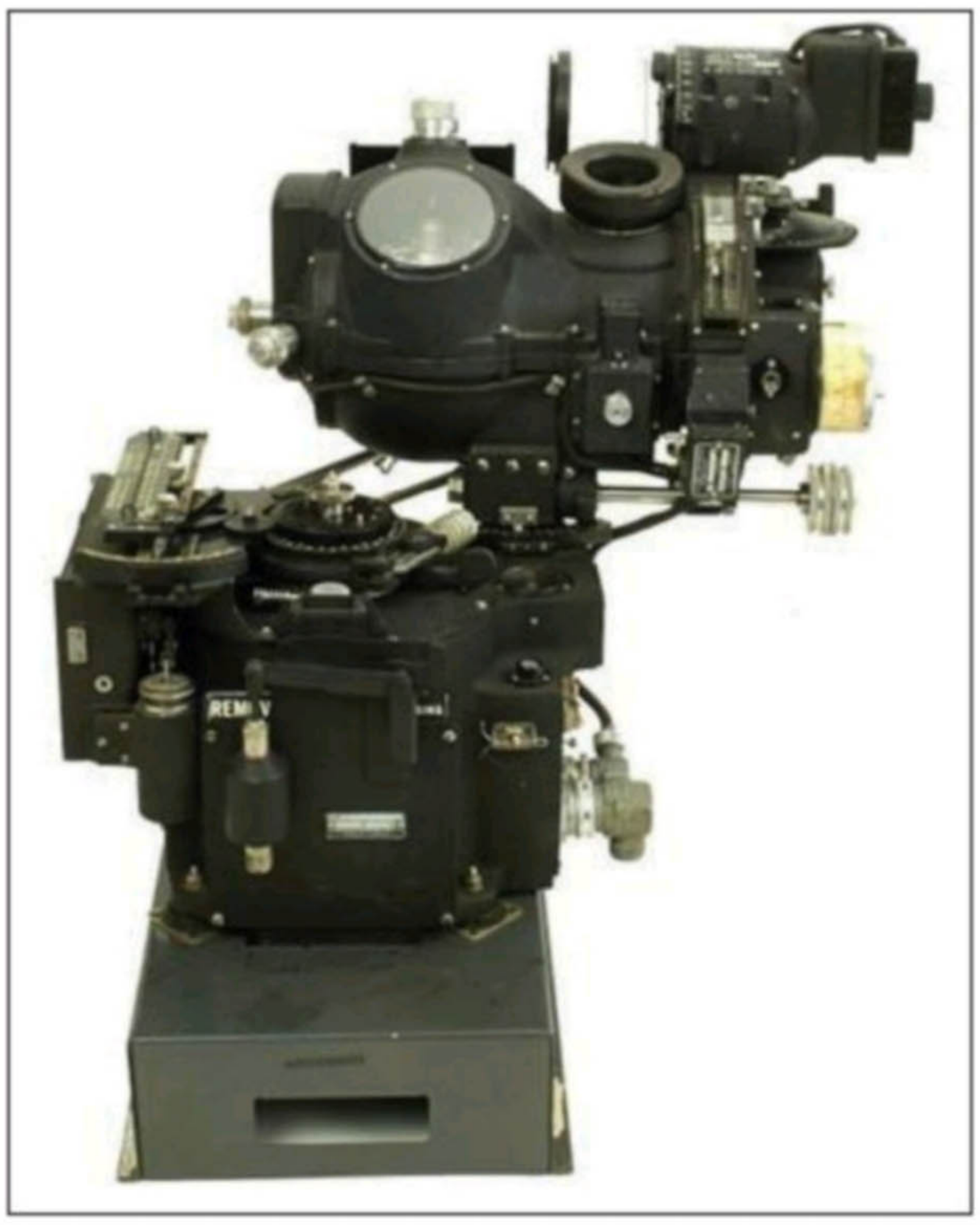


FIGURE 2
The Norden Mk. XV bombsite: an early targeting computer and autopilot controller.

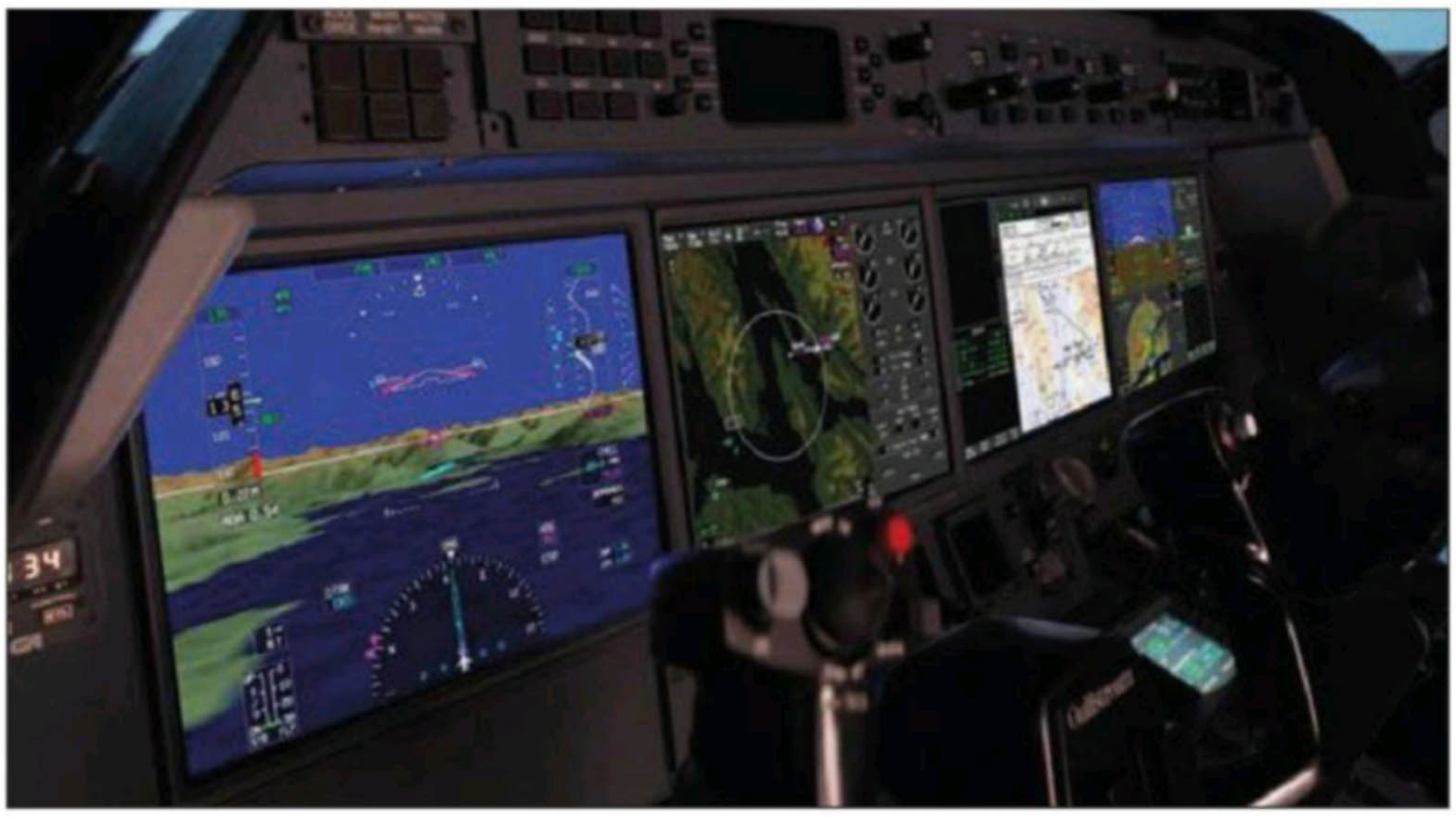


FIGURE 3
Honeywell's SmartView Synthetic Vision System provides a complete 3D rendering of the forward terrain to assist pilots in otherwise low visibility situations.

One scene in *Star Wars* that has always seemed troublesome to me happened during the conclusion: Darth Vader, second in command of the Empire and acclaimed pilot, is about to shoot down Luke Skywalker and prevent the destruction of the Death Star. He has Luke's X-Wing locked in his sights and is about to pull the trigger when all of a sudden, The Millennium Falcon—a civilian ship the size of a small house—blindsides him with a shot from behind, knocking him off course and out of the fight. If even the fighter ship of the lord of the Empire can't detect such an obvious attack, then can any ship in this whole galaxy detect enemy aircrafts? While the evil Empire may not be able to accomplish this feat, we have known of radar since the late 19th century, and have been using it to detect ships and aircraft since the early 20th century. The latest development in this technology, however, has been the integration of AI and machine learning (ML) by RTX's new Raytheon system.

Using a Cognitive Algorithm Deployment System (CADS) integrated into Raytheon's legacy RWR systems for AI and ML processing, Raytheon combines the latest Embedded Graphics Processing Unit with Deepwave Digital's computing stack for developing AI for radio and wireless systems. The result of this is that CADS can deploy special cognitive methods to sense, identify, and prioritize threats in a manner never before seen, with the benefits of increasing air crew survivability, while facilitating the rapid and cost-effective mass deployment of modern AI and ML capabilities. Expected to go into use on USAF jets later this year, specifically on the F-16 Fighting Falcons used by the Air National Guard and the EC-130H Compass Call, but it can be integrated onto any plane using the ALR-69A radar warning receiver, the first digital warning receiver.

In the spirit of the rebel targeting computer heads up display (HUD) Honeywell Aerospace technologies has released the SmartView Synthetic Vision System (SVS) as seen in **Figure 3**. This system, which runs on Honeywell's Primus Epic Integrated Flight Deck, integrates data from multiple onboard databases, GPS and inertial reference systems into a complete 3D rendering of the forward terrain, meaning that pilots no longer have to rely on eyesight, and can see a complete image of their environment, regardless of light or visibility conditions. Improving situational awareness and enhancing operational flexibility increases pilot safety during missions, and simplifies instrument flying in poor visibility conditions.

AI

The integration of artificial intelligence (AI) and embedded electronics is revolutionizing the aerospace and defense industry, driving advancements in both operational efficiency and strategic capabilities. In the aerospace sector, AI is being harnessed to enhance aircraft performance, safety, and maintenance. Embedded systems equipped with AI algorithms can analyze vast amounts of data from sensors and other sources, enabling predictive maintenance, optimizing flight routes, and even automating certain flight tasks. This not only reduces operational costs but also enhances safety by identifying potential issues before they become critical. In the defense sector, AI-powered embedded systems are playing a crucial role in developing autonomous weapons systems, enhancing situational awareness, and improving decision-making processes. These systems can analyze battlefield data in real-time (**Figure 4**), identify threats, and even execute tactical maneuvers with minimal human intervention. Furthermore, the integration of embedded electronics with AI is enabling the development of advanced cybersecurity solutions for both aerospace and defense. These systems can monitor network traffic, identify potential threats, and respond to cyberattacks in real time, safeguarding critical infrastructure and sensitive data. As AI and embedded electronics continue to evolve, their role in the aerospace and defense industry will become even more critical, driving innovation, enhancing security, and shaping the future of warfare and air travel.

Another aerospace use of AI is in air traffic control (**Figure 5**). With an expected 30-fold increase in the volume of air traffic over large cities within the next 15-20 years, especially in the volume of drones and air taxis, much more control will be needed. With AI-driven air traffic control, air traffic can be routed in optimal paths decided by AI, with commands given directly to autonomously piloted vehicles in real time, directly from air traffic control, thus greatly reducing the possibility of accidents during takeoff and landing (the most probable time for accidents to occur.) With the ability to quickly process large datasets in real time and make decisions, AI is perfectly suited for this job.

32-BIT MCU

Further advancing the goal of integrated AI and ML capabilities is the new 32-bit PIC32A family from Microchip (**Figure 6**). This 200MHz family of chips features integrated high-speed analog peripherals, up to 40Msps 12-bit ADCs, high-speed 5ns comparators and 100MHz Gain Bandwidth Product (GBWP) op-amps for intelligent edge sensing so that when paired with a high-performance CPU, multiple functions may be performed on a single MCU. To ensure the safe execution of software, Microchip has integrated hardware safety and security features, such as Error Code Correction (ECC) on Flash and RAM, Memory Built-In Self-Test (MBIST), I/O integrity monitor, clock monitoring, immutable secure boot and Flash access control features.

FIGURE 4

Zero Trust by General Dynamics Information Technology secures communication across vast areas in challenging military tactical environments allowing for real-time execution.

OPTIMIZED
FLIGHT
AI
OPTIMIZED
FLIGHT EARTHS
OPTIMIZED
FLIGHT
OPTIMIZED
FLIGHT PATHS

FIGURE 5

With an expected 30-fold increase in the volume of air traffic over large cities, air traffic control could benefit from AI.

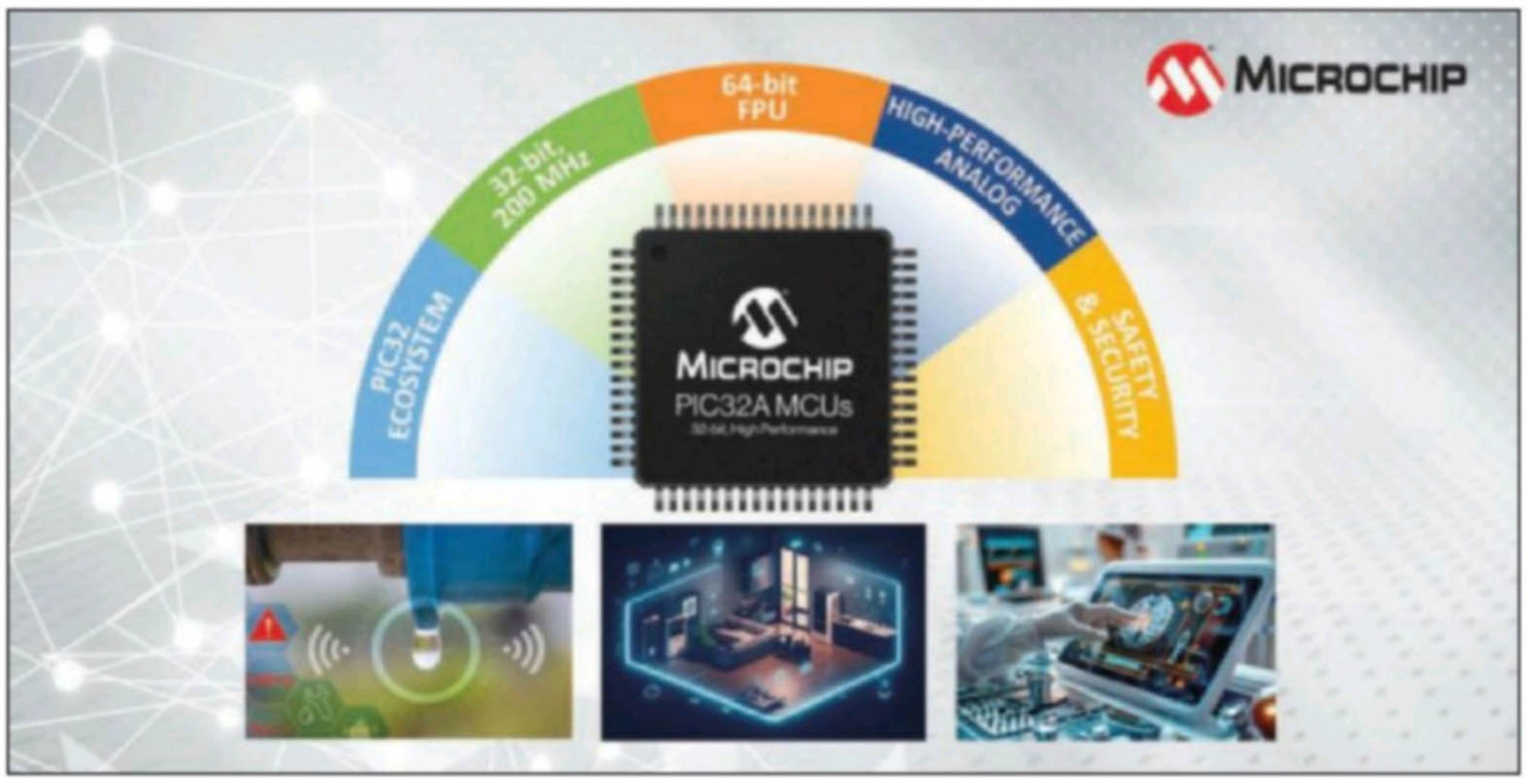


FIGURE 6

The new 32-bit PIC32A family from Microchip designed for enhancing AI and ML capabilities.

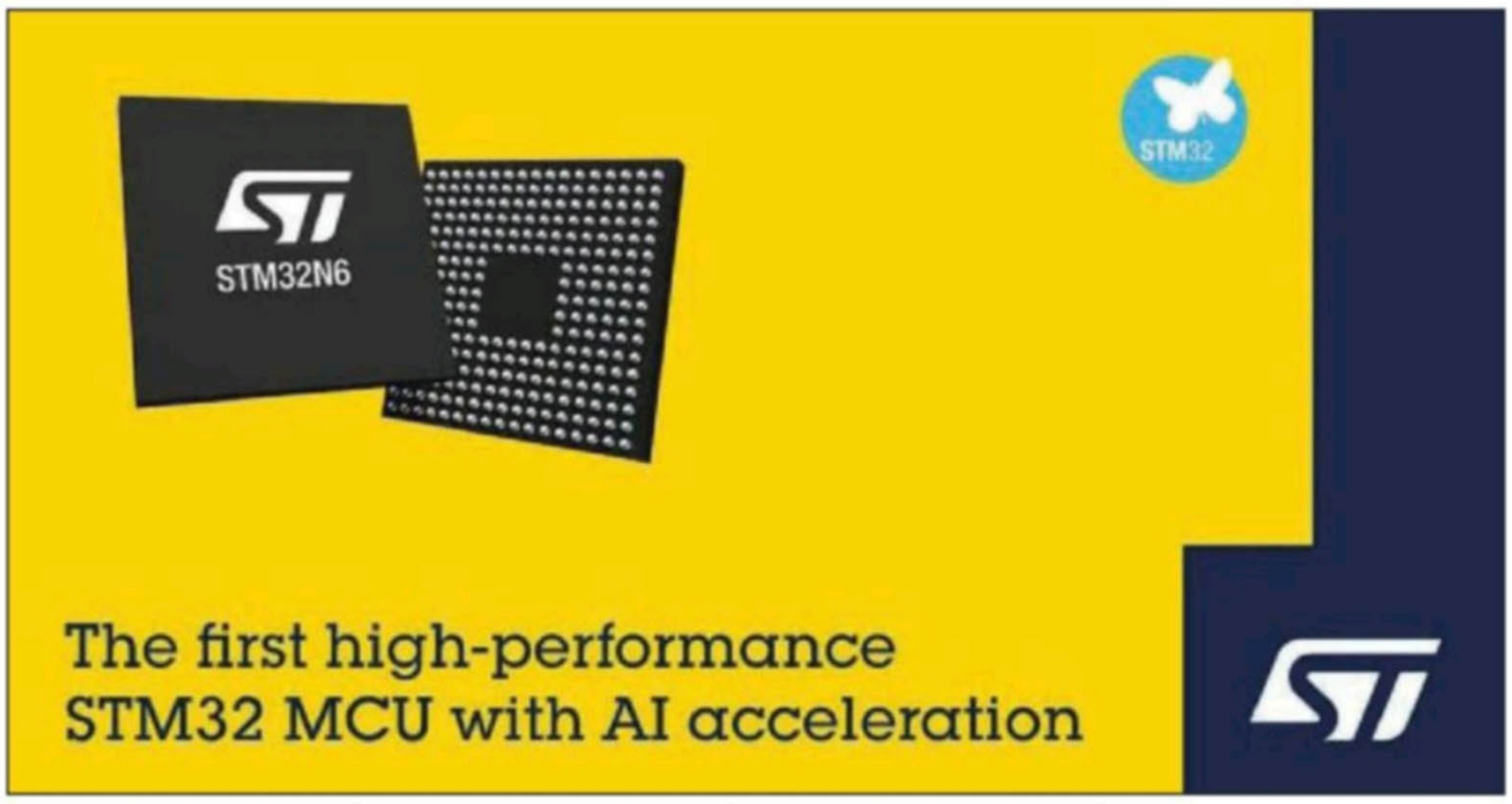


FIGURE 7
STMicroelectronics' STM32 series 32-bit microcontrollers are used in a wide aray of applications.

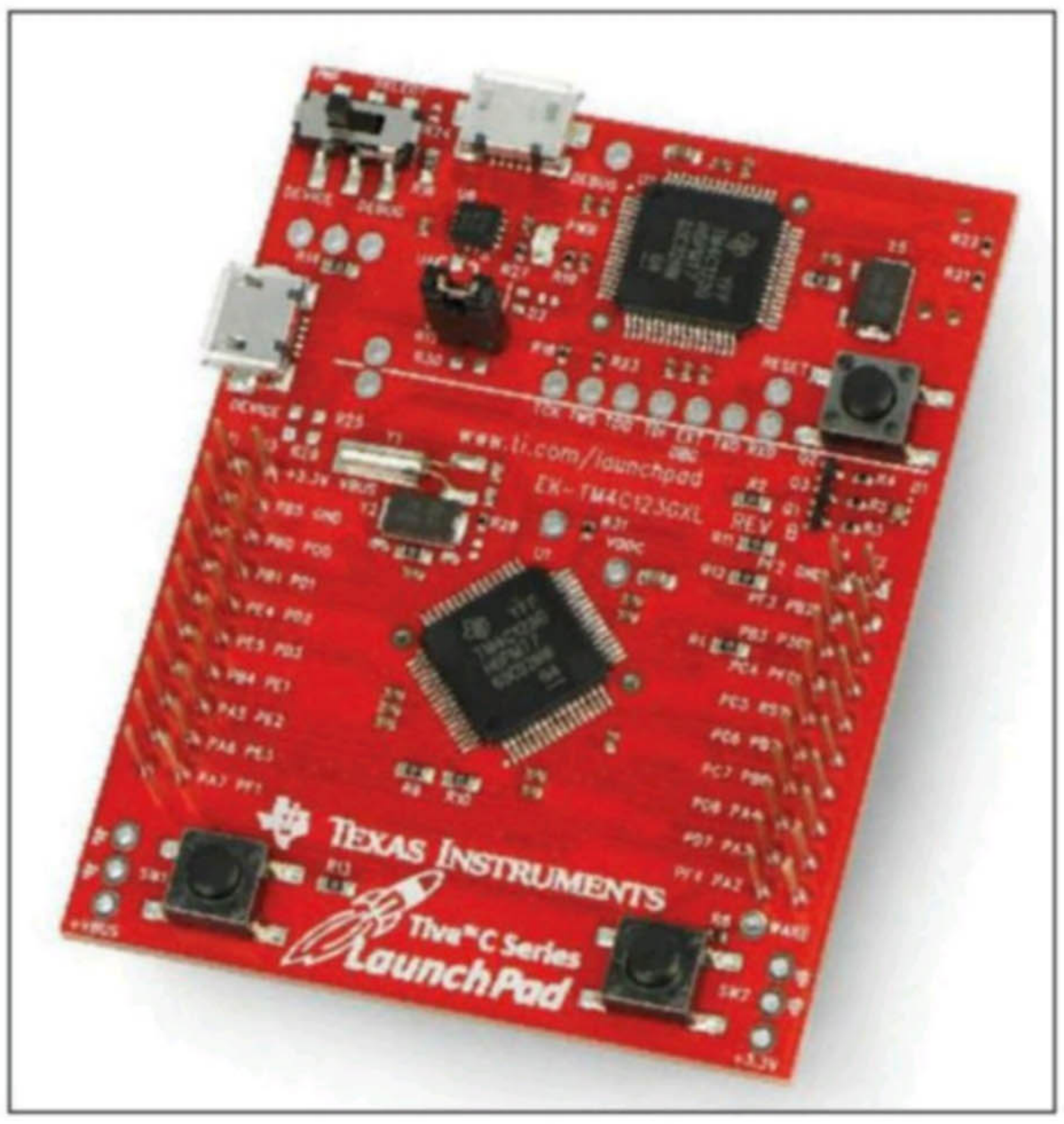


FIGURE 8
Texas Instruments' Tiva C Series microcontrollers for high-performance signal processing and real-time control.

To handle data-intensive math processing applications, a 64-bit floating-point unit (FPU) assists developers in computationally heavy applications that require advanced performance in sensor interfacing and data processing. Not isolated in its usability, the PIC32A MCUs are supported by a number of development tools, including the MPLAB XC32 Compiler, MPLAB Harmony embedded software development framework and the dsPIC33A Curiosity Platform Development Board (EV74H48A) and PIC32AK1216GC41064 General-Purpose DIM (EV25Z08A). With an eye on feature expansion, the Curiosity Development Board provides mikroBUS and Xplained Pro interfaces to connect to Built-In Self-Test Xplained Pro (BIST XPRO) extension kits, sensors, and various Click boards.

STMicroelectronics' STM32 series 32-bit microcontrollers (**Figure 7**) are increasingly prevalent in military applications, valued for their high performance, low power consumption, and extensive ecosystem. Their robust design and availability in various package types and memory configurations allow for flexible integration into diverse systems, from portable handheld devices to larger embedded systems within vehicles or aircraft.

The STM32's proven reliability and adherence to stringent quality standards, including those relevant to military and aerospace, make them a suitable choice for demanding environments where dependability is paramount. Furthermore, the extensive software support, development tools, and readily available third-party libraries simplify the design process, reducing development time and costs while enhancing the overall performance and security of military systems.

Texas Instruments' Tiva C Series microcontrollers offer a compelling alternative (**Figure 8**), particularly where high-performance signal processing and real-time control are paramount. The Tiva C Series, based on the ARM Cortex-M4F processor, provides a powerful FPU crucial for computationally intensive tasks such as advanced signal processing in radar systems or sophisticated guidance algorithms for munitions. Its integrated peripherals, including advanced timers and communication interfaces like SPI and I^{2}C, facilitate seamless integration with various sensors and actuators commonly found in military systems.

Furthermore, the Tiva C Series' robust design, coupled with TI's extensive support for functional safety and security standards, makes it suitable for mission-critical applications requiring high reliability and data integrity. This combination of processing power, integrated peripherals, and robust design makes the Tiva C Series a strong contender for demanding

Additional materials from the author are available at:
www.circuitcellar.com/article-materials

Resources

Deepwave Digital | www.deepwavedigital.com

General Dynamics Information Technology | www.gdit.com

Honeywell Aerospace | https://aerospace.honeywell.com

Infineon | www.infineon.com

Microchip | www.microchip.com

NVIDIA | www.nvidia.com

NXP Semiconductor | www.nxp.com

RTX Raytheon | www.rtx.com/raytheon

STMicroelectronics | www.st.com

Texas Instruments | www.ti.com

Thales | www.thalesgroup.com

Tordex | www.tordex.com

military applications where precise control and high-speed signal processing are essential.

SATCOM

As I mentioned before, the inciting conflict of the *Star Wars* movie is the transfer of a brief video message, and plans for the Death Star. Although above the planet to which the message must be sent, and in easy range for any wireless communication, the message is downloaded to a hard disk and hand delivered by two droids. I have often thought that they would have benefited from some form of satellite communication: the ability to send information in a straight path to a point in orbit around a planet, which acts as a repeater for the message as seen in **Figure 9**.

Thales has been working on just that with its secure multi-orbit Low Earth Orbit (LEO), Medium Earth Orbit (MEO), High Earth Orbit (HEO), and Geostationary Orbit (GEO), and its multi-band SatCom communication terminals. With end-to-end encryption, and anti-jamming measures, this satellite communication is perfect for mission critical data to be sent and received securely, confidentially, and uninterrupted, even in the most remote of environments, thus eliminating the risks and inefficiencies that come with older forms of communication such as radio and telecommunications, not to mention two inept droids.

ROBOTICS

Today, many sectors of the Aerospace industry incorporate robotics, enhancing efficiency, reducing safety concerns, and improved consistency. Robotic welding, drilling, fastening, and painting are just a few of the roles robotics improve the automation process. Sensors technology has advanced to the point where mission critical inspections are preformed by robots (**Figure 10**).

One of the more overlooked parts of aerospace and defense technology is the operation and failure of supply chains. When a supply chain fails, it creates cost overruns and delays, and sometimes prevents mission critical equipment from getting to where it needs to go. Unlike the highly connected automotive industry, the aerospace and defense industries lack the cooperation to manage and fix supply chain interruptions quickly and easily. For this reason, preventing supply chain interruptions is even more critical. By using general-purpose collaborative robots (cobots) that can work alongside humans on factory floors and in production lines, trackable automation can be introduced at a much lower cost, as retooling the production process will not be required. These cobots, driven by AI, can adapt to new tasks with simple instructions, and do not require the same intense level of programming required by traditionally programmed robots.

FINAL TRANSMISSION

While much of the technology seen in this classic movie is still in the realm of science fiction—I'm still waiting on my lightsaber—we have caught up to, and in many ways surpassed what has been shown to be the technological capabilities of this galaxy far, far away. With Thales SatCom technology, we can send and receive data across great distances to even high orbits safely and securely, thus eliminating the need for droid data carriers. With Honeywell's SmartView Synthetic Vision System, we can produce better renderings of terrain than simple graphics of rebel targeting computers. The capabilities of RTX Raytheon's CAD System embarrass the complete lack of warning systems seen on Imperial fighter crafts. In less than 50 years, we have surpassed a whole generation's ideas of science fiction through the same dedication and imagination that first sent man into the sky on spruce and cotton.

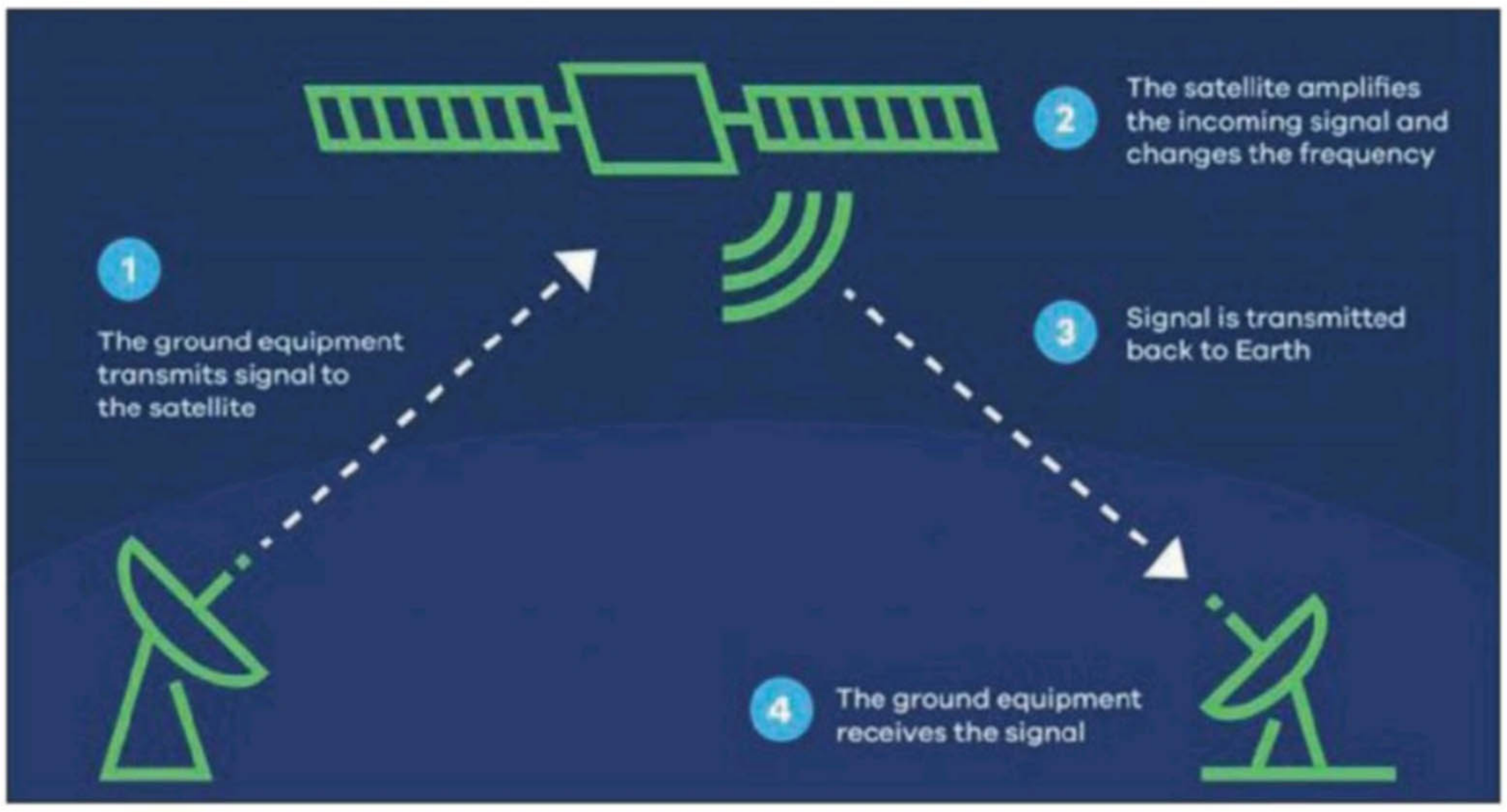
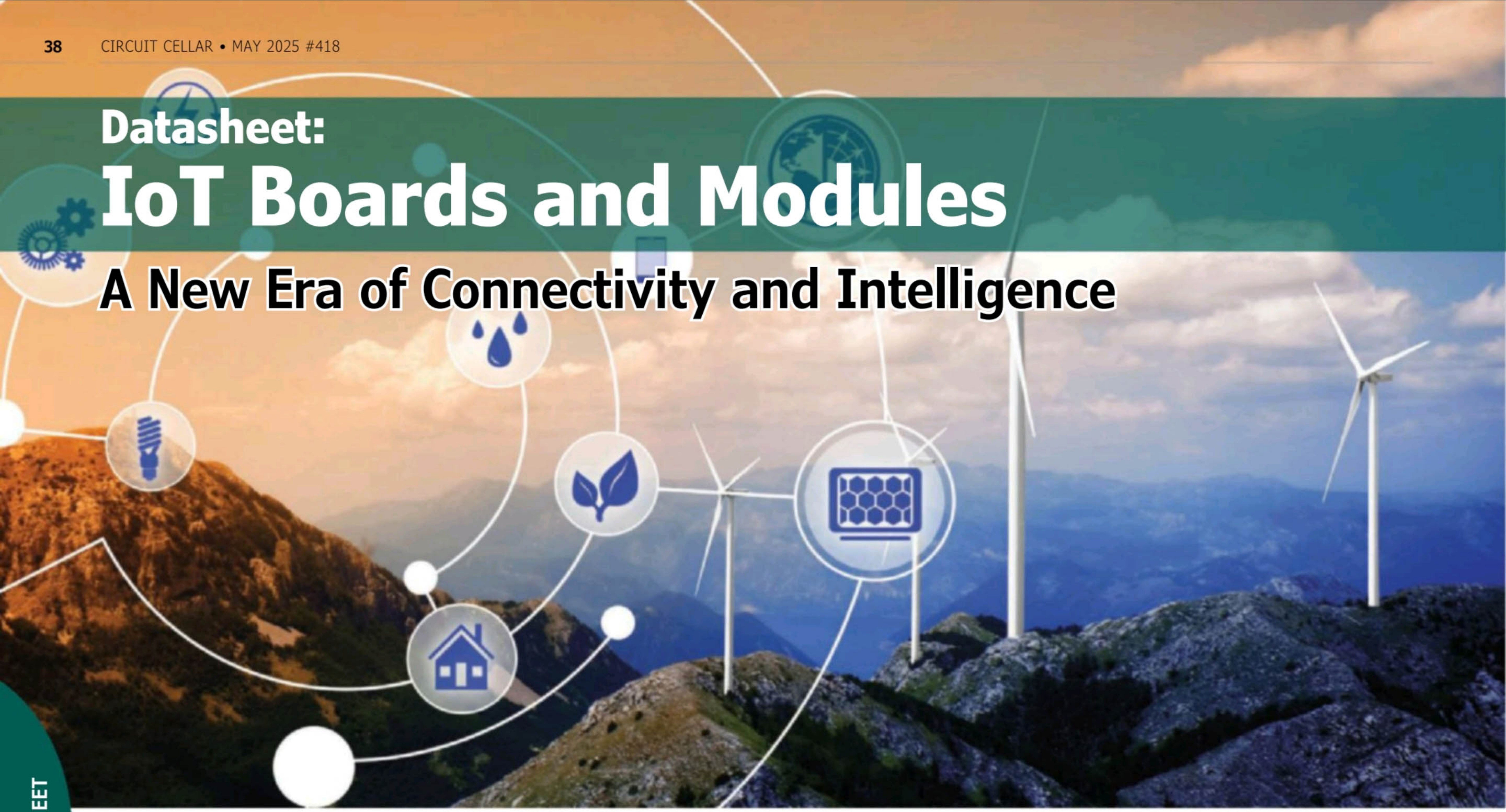

1
The ground equipment transmits signal to the satellite
2
The satellite amplifies the incoming signal and changes the frequency
3
Signal is transmitted back to Earth
4
The ground equipment receives the signal

FIGURE 9
Satellite communications bounces signals to and from a satellite for uninterrupted communication.

FIGURE 10
Aerospace robotics automate the production process.

Datasheet:
IoT Boards and Modules
A New Era of Connectivity and Intelligence



Dive into IoT board advancements. Breakthroughs in miniaturization, processing power, security, and energy efficiency are transforming design and deployment strategies, revolutionizing the IoT landscape.

The pace of change in the Internet of Things (IoT) space has always been dynamic outpacing most predictions. For those of you who've seen the evolution from 8-bit microcontrollers to today's sophisticated system-on-chips (SoCs), the advancements in IoT boards are particularly striking. They have moved beyond the simple sensor-to-cloud paradigm, entering an era of sophisticated edge processing, demanding new levels of integration and performance.

One of the most significant trends is the convergence of technologies. No longer are we dealing with isolated components. Today's IoT boards seamlessly integrate multiple wireless protocols—Wi-Fi, Bluetooth, LoRaWAN, cellular—often on a single chip. This simplifies design, reduces board space, and lowers power consumption, all critical concerns for battery-powered applications. This integration isn't just about connectivity; it extends to processing capabilities as well.

The rise of AI at the edge is fundamentally altering the design considerations for IoT boards. Powerful, low-power processors with dedicated hardware accelerators for machine learning are becoming increasingly common. This allows for real-time data analysis and decision-making directly on the device, reducing latency and dependency on cloud services. We're no longer just collecting data; we're processing it intelligently, enabling sophisticated applications like predictive maintenance, anomaly detection, and autonomous operation.

Security, always a paramount concern, is being addressed with more sophisticated hardware-based security features. Secure boot, trusted execution environments, and hardware-level encryption are becoming standard features, mitigating the risks associated with deploying devices in potentially hostile environments. This isn't just about protecting data; it's about ensuring the integrity and reliability of the entire system.

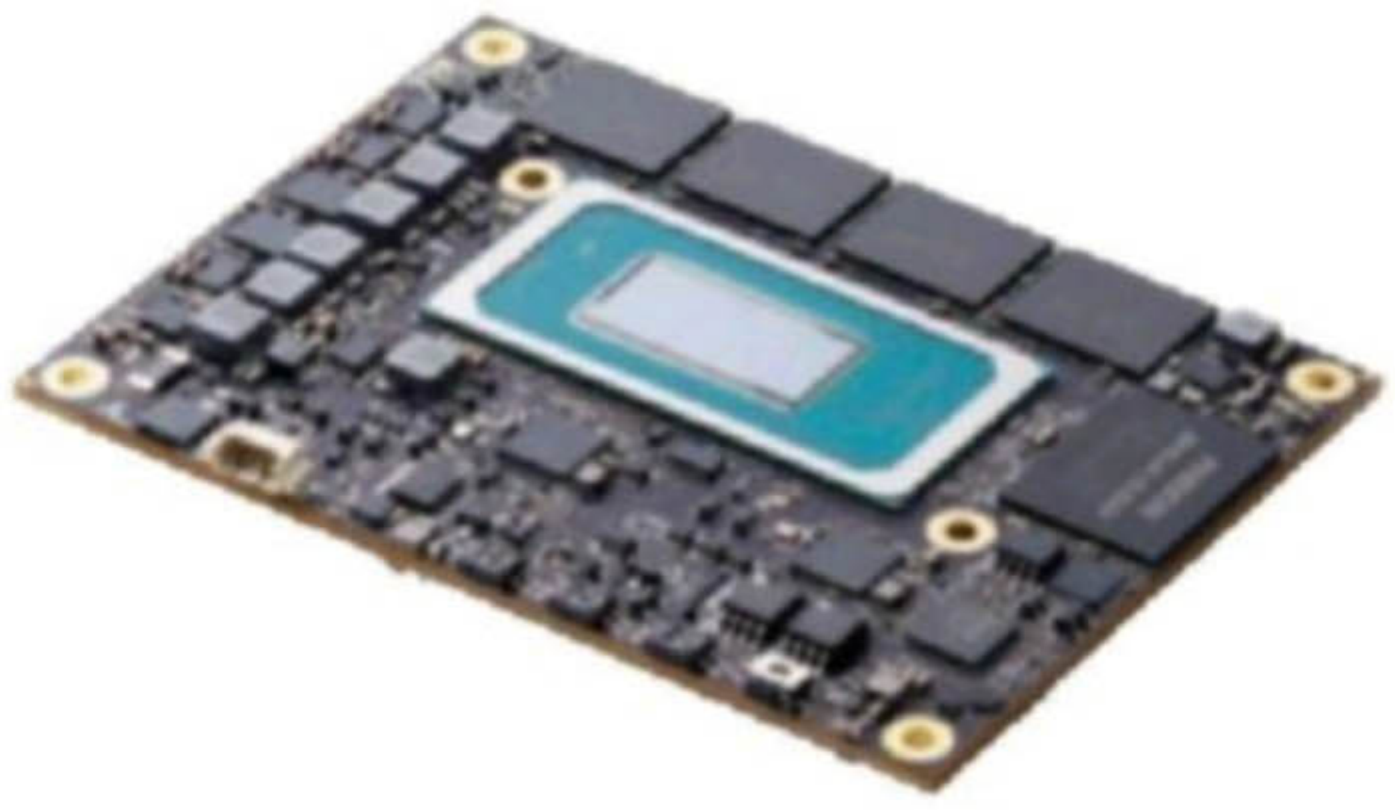
Power efficiency continues to be a major driver of innovation. Advances in low-power processors, energy-harvesting techniques, and efficient power management ICs are extending the operational life of battery-powered IoT devices. This is crucial for applications deployed in remote locations or where battery replacement is difficult or impractical.

Edge IoT connectivity is revolutionizing remote deployments. Reliable, low-power wide-area networks (LPWANs) such as LoRaWAN and NB-IoT are enabling communication in previously inaccessible locations. This opens up exciting possibilities for environmental monitoring, smart agriculture, and industrial automation in remote and challenging environments. The focus is shifting toward robust, resilient designs capable of operating reliably under harsh conditions with minimal maintenance.

The design process itself is also undergoing a transformation. The availability of comprehensive software development kits (SDKs), cloud-based development platforms, and pre-certified modules is accelerating development cycles and reducing time to market. This allows engineers to focus on the unique aspects of their applications rather than getting bogged down in low-level details.

Ingenuity and a relentless pursuit of innovation within the field have driven the evolution of IoT boards. This convergence of connectivity, processing power, security, and power efficiency is creating a new generation of intelligent, connected devices poised to transform industries and shape our future. The challenges remain—managing complexity, ensuring security, and optimizing for power—but the tools and technologies at our disposal are more powerful than ever before.

NEXT MONTH'S TOPIC: AC-DC Power Supplies Send related product announcements to editor@circuitcellar.com



Rugged Small Form

COM-HPC-mMTL from AdLink sets a new standard for miniaturized, high-performance embedded computing. Its small form factor is coupled with high bandwidth and power efficiency, making it ideal for space-constrained applications demanding significant processing power. Key distinctions include support for high-performance CPUs and GPUs, advanced interfaces like PCIe Gen 5, and robust thermal management. Designed for demanding applications such as AI at the edge, industrial automation, and high-performance computing in small form factors. The standard's focus on scalability and interoperability ensures flexibility in system design and component selection.

- Intel Core Ultra Processor
- Up to 14 CPU cores (6 P-cores, 8 E-cores and 2 LP E-cores)
- 8 Xe GPU cores for enhanced graphics
- NPU Built-in Neural Processing Unit for AI and machine learning tasks
- Memory Supports up to 64GB LPDDR5x memory
- PCIe Lanes: Up to 16 PCIe Gen 5 lanes for fast data transfer and connectivity
- Ethernet: Dual 2.5GbE Ethernet ports for high-speed networking
- Power Input: Single power rail input with a voltage range of 8V to 20V
- Power Consumption: Delivers up to 107W of power
- Storage Interfaces: Supports multiple SATA and USB4/USB3/2.0 interfaces
- Advanced thermal management
- Wide operating temperature range

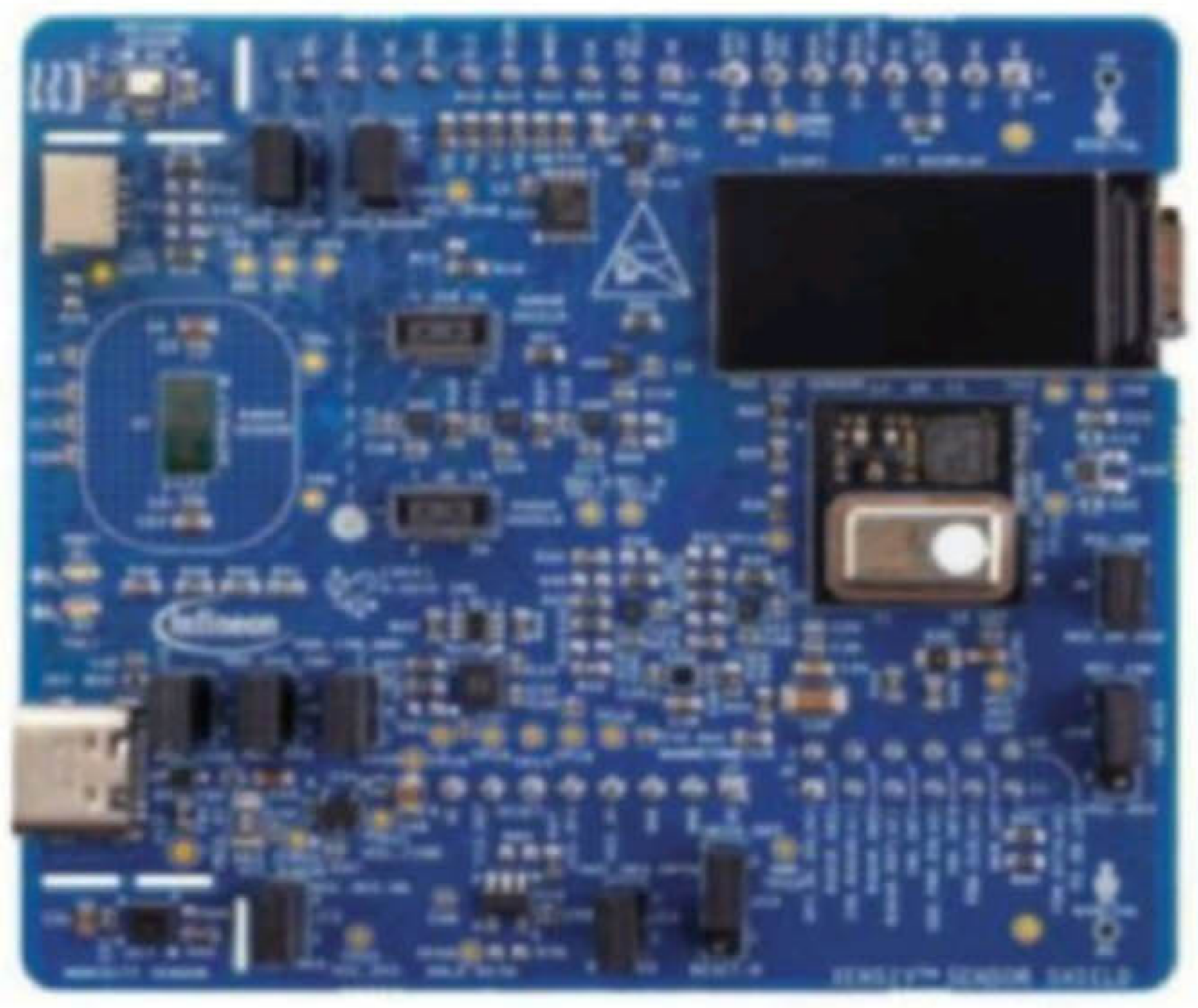
ADLINK
www.adlinktech.com

Global Networking

The Digi XBee3 Global LTE Cat 4 module stands out for its robust performance and global connectivity in a compact form factor. With wideband LTE Cat 4 support, enabling high-speed data transmission across various global networks. Its integrated GPS functionality simplifies location tracking, while its secure design and multiple interface options enhance application versatility. The module's ease of integration with existing XBee ecosystems and its low power consumption make it a cost-effective solution for various IoT applications. Its rugged design ensures reliable operation in challenging environments.

- Supports LTE Cat 4 with download speeds up to 150Mbps
- Operates on various global LTE bands including Band 2, 3, 4, 5, 7, 8, 12, 13, 17, 20, 28, 38, 40, 41
- Integrated GPS receiver for location tracking with accuracy up to 5 meters
- UART, SPI, and I²C interfaces
- Secure boot and encryption
- Low power consumption, typically drawing less than 1.5A at peak
- Measures approximately 25mm x 30mm x 5mm
- XBee Ecosystem Integration
- Built to withstand harsh environments with operating temperatures ranging from -40°C to 85°C
- Wideband LTE Support
- High-Speed Data Transmission
- Integrated GPS functionality
- Enhanced security features

DIGI
www.digi.com



Precise Data Collection

The XENSIV Sensor Shield by Infineon Technologies is a versatile platform designed for rapid prototyping and development of sensor-based applications. It features a range of advanced sensors, including a 60GHz radar for motion detection, a 5V CO_2 sensor utilizing photoacoustic spectroscopy, and various environmental sensors for temperature, humidity, and pressure. This shield stands out for its compatibility with Arduino platforms, making it easy for engineers to integrate and test multiple sensor functionalities in a compact form factor, enhancing the development of IoT and smart applications.

- 60GHz radar for precise motion detection
- 5V CO_2 sensor based on photoacoustic spectroscopy
- Integrated temperature sensor
- Humidity sensor with high sensitivity and accuracy
- 3-axis magnetometer sensor
- Pressure sensor
- MEMS microphones for sound detection and analysis
- IMU accelerometer for motion tracking
- TFT display
- Compatible with Arduino platforms
- Compact design
- Low power consumption
- Supports various communication interfaces including I²C and SPI
- Operating temperature range from -40°C to 85°C for harsh environments
- High signal-to-noise ratio

Infineon
www.infineon.com

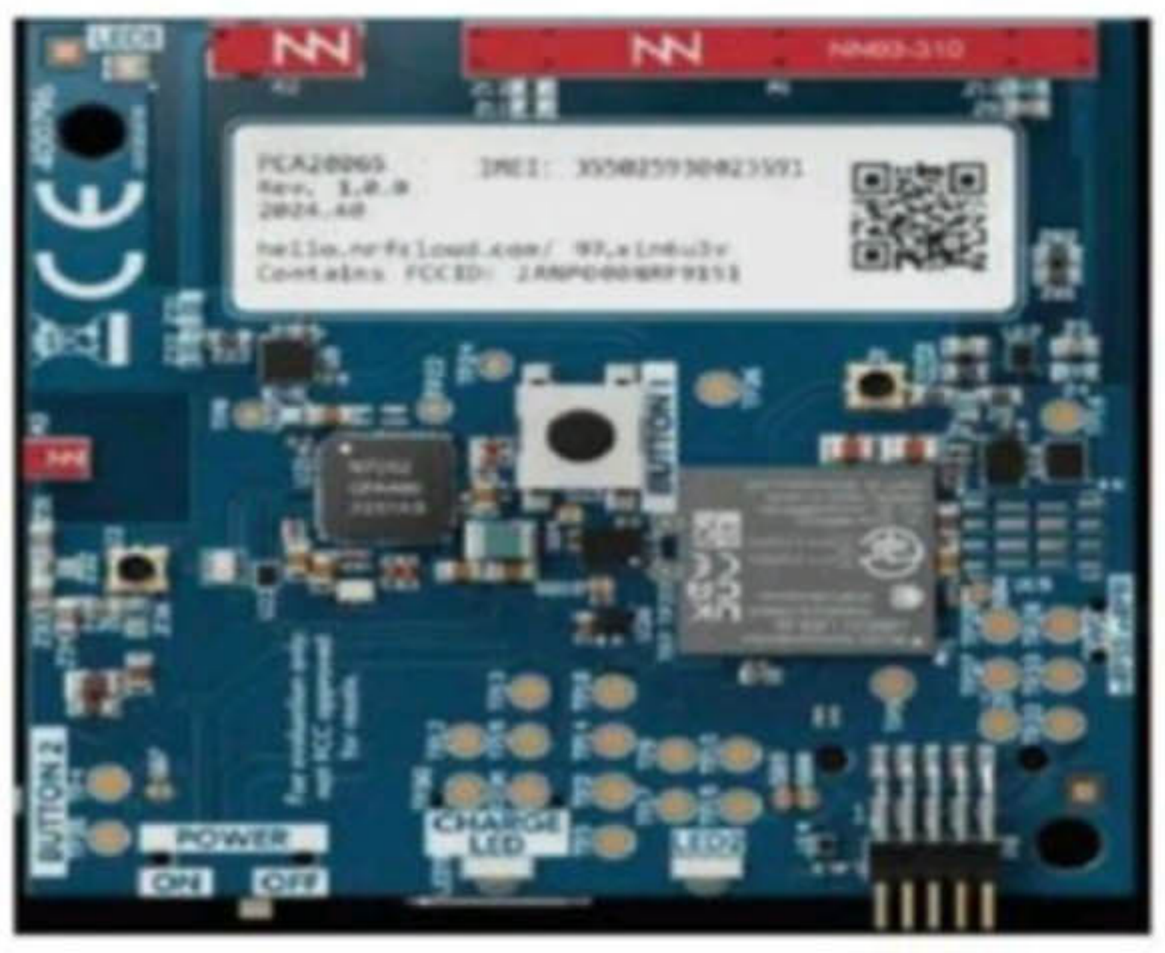
DATASHEET URLS:

ADLINK: www.adlinktech.com/Products/Computer_on_Modules/COM-HPC_Mini_Module/COM-HPC-mMTL

DIGI: www.digi.com/resources/library/data-sheets/digi-xbee-3-global-lte-cat-4-development-kit-datas

Infineon: www.infineon.com/dgdl/Infineon-SHIELD_XENSIV_A_UG-UserManual-v02_00-EN.pdf?fileId=8ac78c8c93956f5001939dcf496b4bc4

Datasheet:
IoT Boards and Modules



Rapid Prototyping

The Nordic Thingy:91 X is a versatile IoT prototyping platform designed for seamless global connectivity. It supports LTE-M, NB-IoT, and Wi-Fi, making it ideal for a wide range of applications. Key features include an integrated nRF5340 system-on-chip (SoC), various environmental sensors, and a rechargeable Li-Po battery. Its compact design and low power consumption make it perfect for battery-operated devices. The platform is user-friendly, facilitating rapid development and prototyping, which is essential for engineers looking to innovate in IoT solutions.

- Supports LTE-M and NB-IoT for global cellular connectivity
- Integrated dual-band 2.4/5GHz Wi-Fi
- Powered by the nRF5340 SoC
- One dual-band antenna
- Rechargeable Li-Po battery with a capacity of 1350mAh
- Nano/4FF SIM card slot supporting (e) SIM with 10 MB preloaded data
- User-programmable buttons and LEDs
- Integrated 6-axis IMU with gyroscope for precise motion tracking
- Low-power 3-axis accelerometer
- Multiple sensors for measuring temperature, humidity, and pressure
- USB Interface
- Preloaded SIMs
- Compact form factor
- Optimized for low power consumption
- Comprehensive software support for rapid development and prototyping
- High signal-to-noise ratio for reliable data transmission

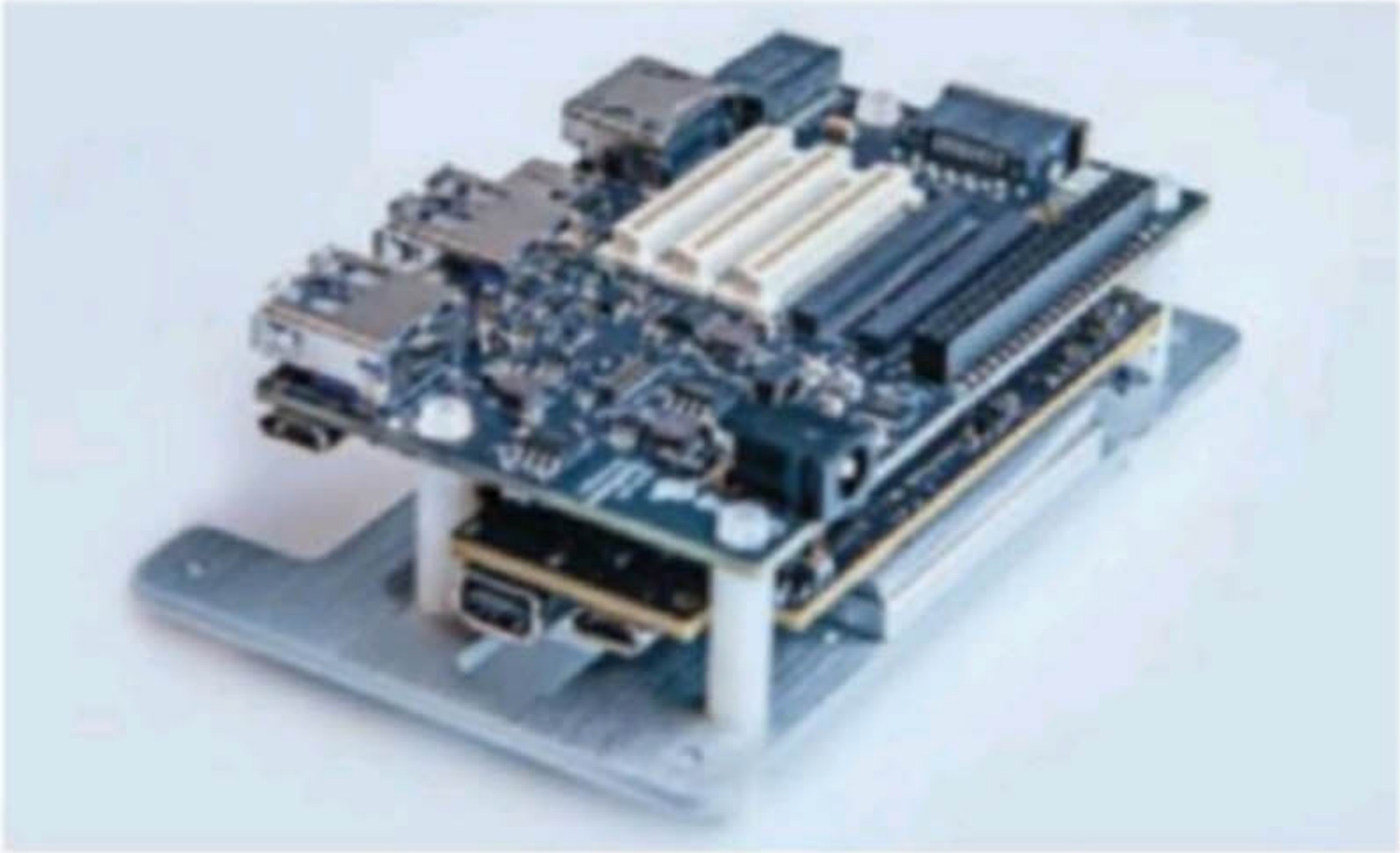
Nordic Semiconductor
www.nordicsemi.com

AI Performance

The Jetson Orin Nano is a powerful AI computing platform designed for edge applications by NVIDIA. Delivering up to 40 TOPS of AI performance in a compact form factor. It features a 6-core ARM Cortex-A78AE CPU and an NVIDIA Ampere GPU, optimized for energy efficiency and high-speed processing. With support for various AI frameworks and extensive software tools, it enables engineers to develop advanced robotics, computer vision, and deep learning applications efficiently. Its low power consumption and robust design make it ideal for deployment in diverse environments.

- Delivers up to 40 TOPS of AI performance
- 6-core ARM Cortex-A78AE processor
- NVIDIA Ampere architecture optimized for AI workloads
- 8GB of high-speed LPDDR5 RAM for
- microSD card for expandable storage
- Operates between 7W and 15W
- Gigabit Ethernet port for high-speed network access
- 40-pin expansion header for additional peripherals
- USB-C port for data transfer and power supply
- Supports HDMI for display connectivity
- MIPI CSI-2 interface for cameras
- Runs on NVIDIA Jetson Linux
- Comprehensive software stack including TensorRT and CUDA
- Integrated thermal management
- Hardware-based security

NVIDIA
www.nvidia.com



Innovative Connectivity

The RB3 Gen 2 Developer Kit from Qualcomm is a cutting-edge platform designed for IoT and AI applications, offering enhanced performance and connectivity. It features the Qualcomm Kryo 670 CPU and a Hexagon processor with a fused AI accelerator, enabling real-time processing for advanced applications like smart cameras and autonomous systems. With support for Wi-Fi 6E, 5G, and various I/O interfaces, this kit is ideal for developers looking to create innovative, connected devices with high-speed data handling and low latency.

- 2x Cortex-A78 at 2.1GHz and 4x Cortex-A55 at 1.8GHz
- Adreno A642L at 315MHz
- 2x HVX cores with ~3.5 INT8 TOPS
- 8GB LPDDR4x RAM
- Supports microSD
- Multi-gigabit Wi-Fi 6E with speeds up to 3.6Gbps
- Supports 5G for high-speed mobile
- Integrated Bluetooth 5.2 for short-range communication
- Advanced ISPs for single or multiple camera setups
- Supports FHD+ at 120Hz for high-resolution displays
- High-quality audio processing
- Multiple standard expansion connectors
- Efficient power consumption
- Modular design based on the 96 Board compact specifications
- Comprehensive software support for rapid prototyping
- Hardware-based security
- Integrated cooling solutions

Qualcomm
www.qualcomm.com

DATASHEET URLS:

Nordic Semiconductor: www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/Nordic-Thingy91-X-PB-1.0.pdf

NVIDIA: https://nvdam.widen.net/s/zkfqjmtds2/jetson-orin-datasheet-nano-developer-kit-3575392-r2

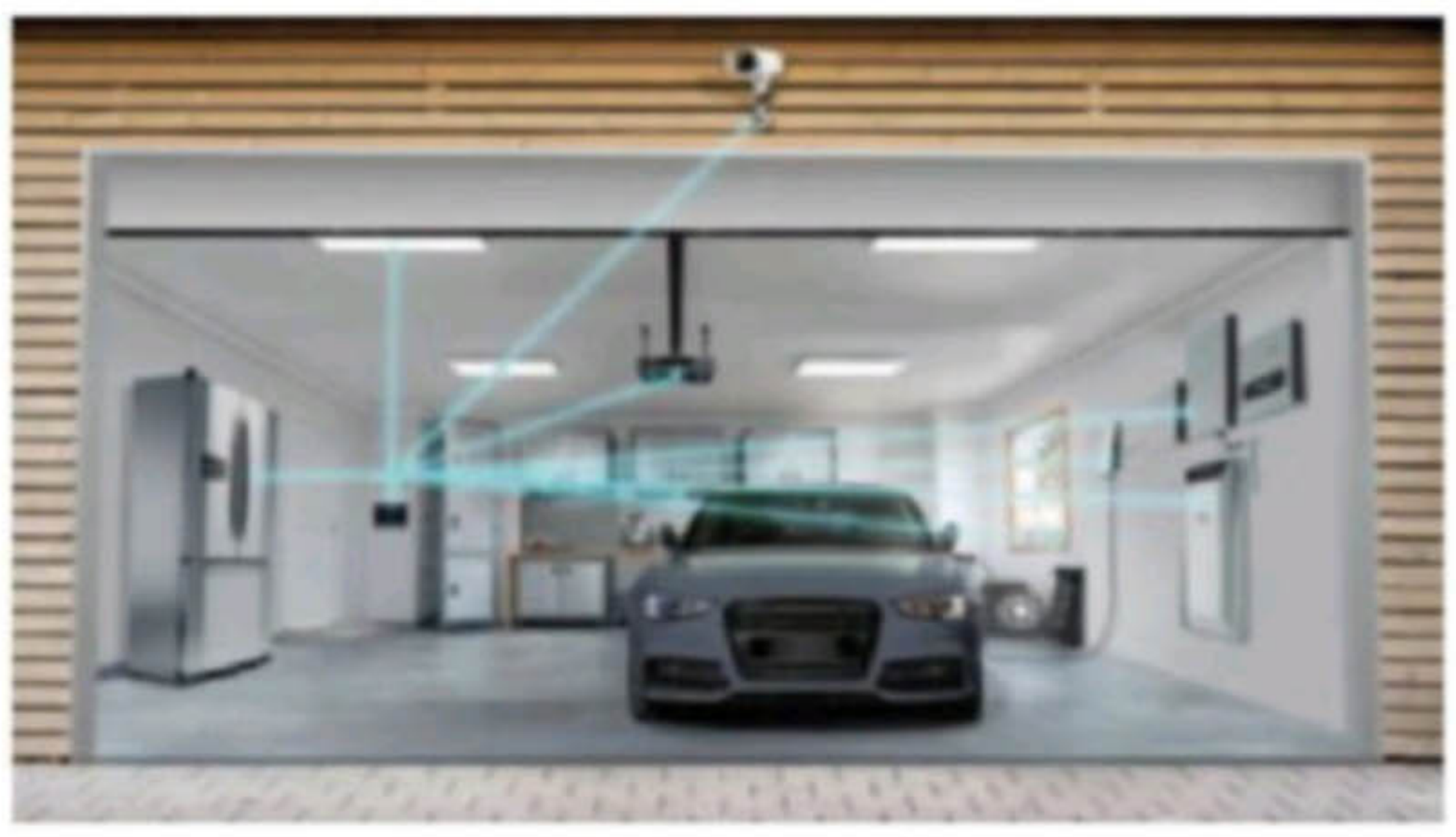
Qualcomm: www.qualcomm.com/developer/hardware/rb3-gen-2-development-kit

Efficient Complexity

The STM32WBA6 is a versatile wireless microcontroller designed for IoT applications, featuring a powerful ARM Cortex-M33 core. It supports multiple wireless protocols, including Bluetooth Low Energy (BLE) and Zigbee, making it ideal for smart devices such as wearables and home automation systems. With up to 2MB of flash memory and 512KB of RAM, it can handle complex applications efficiently. Its low power consumption ensures long battery life.

- ARM Cortex-M33 efficient processing
- Up to 2MB flash memory
- Up to 512KB RAM
- Supports Bluetooth LE, Zigbee, Thread, and Matter Wireless Protocols
- Operates at 2.4GHz radio frequency
- Low power consumption
- USB OTG support
- Hardware-based security
- Multiple GPIOs for peripheral connections
- Integrated analog-to-digital converter for sensor interfacing
- Multiple timers for precise control
- Pulse-width modulation support for motor control
- Direct memory access
- Operating temperature from -40°C to +105°C
- Available in compact packages
- Comprehensive software libraries for rapid prototyping

STMicroelectronics
www.st.com



Integrated Wi-Fi 6

The CC3301MOD SimpleLink is a highly integrated Wi-Fi 6 and Bluetooth Low Energy (BLE) module designed for embedded applications from Texas Instruments. It offers robust connectivity with low power consumption, making it ideal for IoT devices. The module supports advanced features such as secure connectivity and is compatible with various host processors, enabling seamless integration into diverse applications. It's an affordable and reliable choice.

- Supports Wi-Fi 6 (802.11ax)
- Bluetooth Low Energy 5.4
- Companion module for any processor or MCU host capable of running a TCP/IP stack
- Optimized for low power usage
- Supports advanced security protocols for secure data transmission
- Integrated antenna options
- Operates at 3.0V to 3.6V for versatile power supply options
- Designed to operate in temperature range from -40°C to +85°C
- Includes 512KB of RAM for efficient data handling
- Up to 4MB of flash memory
- Multiple GPIOs for connecting various peripherals
- Comprehensive software development kit for rapid prototyping
- Compact design
- Supports data rates up to 1.2Gbps
- Complies with regulatory standards for wireless communication
- SPI and UART interfaces

Texas Instruments
www.ti.com

Secure and Rugged

The XPedite7770 3U VPX is a high-performance embedded single board computer from X-ES. Based on the Intel Xeon D-1700 Processor. It features up to 48GB of DDR4 ECC memory and offers advanced networking capabilities with 100-Gigabit Ethernet support. Designed for rugged environments, it incorporates SecureCOTS technology for enhanced security. This module is ideal for demanding applications in military and aerospace sectors, providing robust processing power and extensive I/O options for versatile deployment in edge computing scenarios.

- Intel Xeon D-1700 series
- Up to 48GB of DDR4 ECC SDRAM in three channels
- Up to 32GB of onboard SLC NAND flash
- Supports 100GBASE-KR4, 10GBASE-KR, and 10/100/1000BASE-T
- Multiple USB 2.0 and serial ports
- 3U VPX compliant for rugged applications
- Designed for extreme temperature ranges
- Supports a wide range of input voltages for flexibility
- Incorporates SecureCOTS
- Multiple PCIe lanes
- Advanced thermal management
- Compact design suitable for space-constrained applications
- Integrated debugging features
- Meets military and aerospace standards
- Advanced networking capabilities

X-ES
www.x-es.com

DATASHEET URLS:

STMicroelectronics: www.st.com/resource/en/datasheet/stm32wba62ci.pdf

Texas Instruments: www.ti.com/lit/gpn/cc3301mod

X-ES: https://www.xes-inc.com/products/sbcs/xpedite7770

Embedded System Essentials

Experimenting with CHERI on the Sonata Board

By
Colin O'Flynn

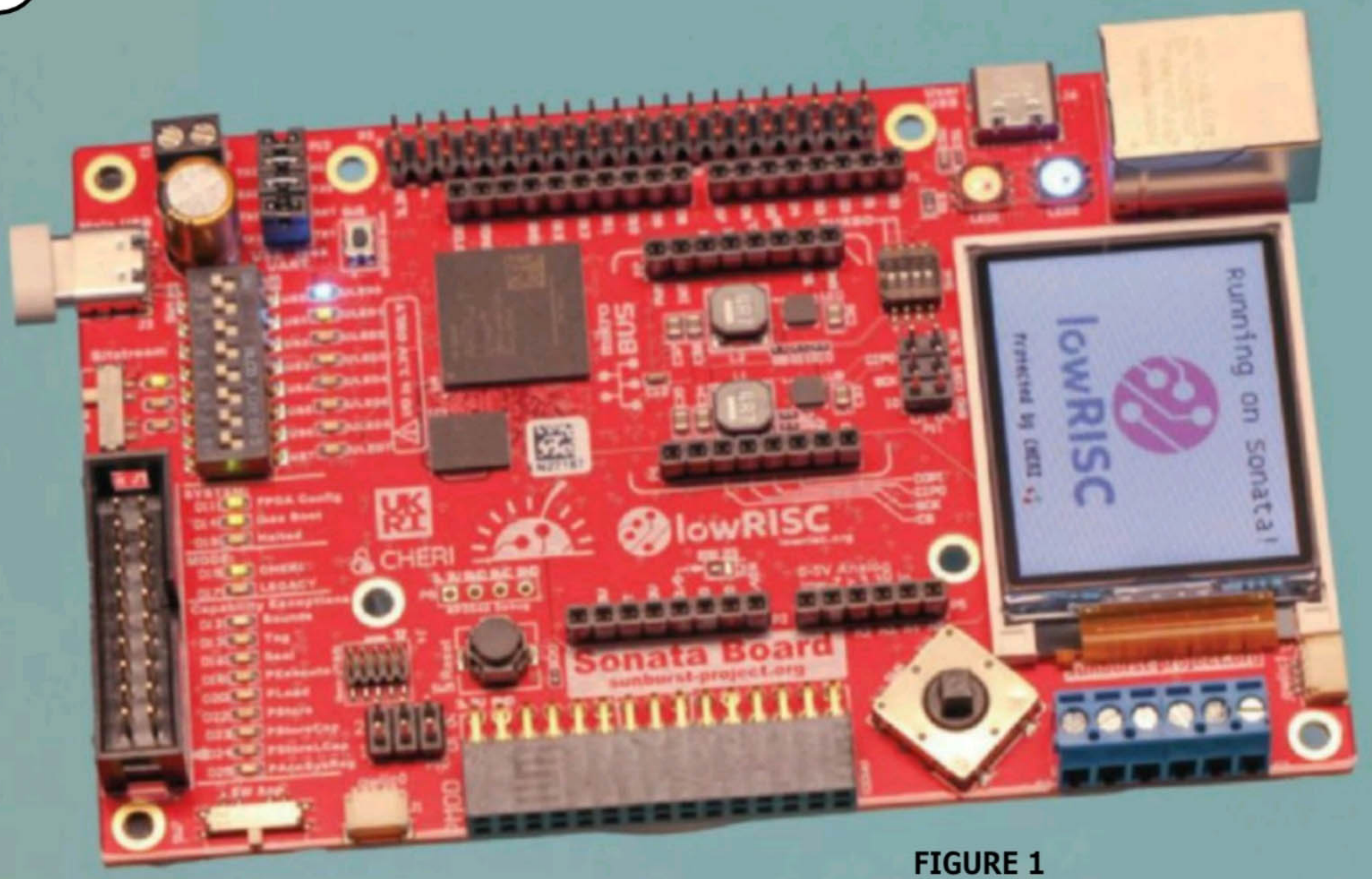


FIGURE 1
An overview of the Sonata Board

The CHERI extension for RISC-V provides a promising way of designing fundamentally secure systems. I introduced this back in November 2023, but at the time there was no easy hardware to work with. This article introduces the open-source hardware (OSHW) Sonata board, which lets you experiment with CHERIoT on a more user-friendly platform.

Back in my November 2023 article in Issue #400 of *Circuit Cellar* I introduced you to the CHERI and CHERIoT architecture, which I presented as a promising way of securing your existing C/C++ code with minimal effort. At the time, I lamented that it was difficult to "get started" with CHERIoT, as it required you to build your own soft-core processor.

Since then, the Sunburst Project has released a board with prebuilt FPGA images, called the Sonata Board. While it's still not as cheap as a simple microcontroller development kit, it has a similar development flow, only requiring you to compile code for the target. The project provides reference code, and the physical platform can be expanded with a variety of add-on boards (shields).

In this article, I'll quickly remind you of what CHERIoT is, before showing you how you could use this with the physical system. The physical board is something I helped design, but the real effort behind Sonata is the FPGA work and the reference platform, which was developed by a team at lowRISC. The physical board is fully open-source, so you can build your own or modify it for specific tasks as you see fit.

WHAT'S CHERI ALL ABOUT?

CHERI (Capability Hardware Enhanced RISC Instructions) is a computer architecture that brings fine-grained memory safety and isolation to hardware. It works by associating capabilities (pointers with bounds and permissions) to every memory reference, preventing programs from accessing memory they shouldn't and enabling strong compartmentalization. In practice, this mitigates vulnerabilities such as buffer overflows and use-after-free bugs that account for a large share of security issues in C/C++ software.

CHERIoT (Capability Hardware Extension to RISC-V for Internet of Things) is a variant of CHERI designed for microcontrollers and small embedded systems. CHERIoT builds on CHERI and RISC-V to define a new 32-bit ISA and a microcontroller-focused RTOS that provides object-level spatial memory safety, deterministic use-after-free protection, and lightweight compartmentalization directly in the C/C++ programming model. If you want more details of this implementation, refer to the resources of this article, or my article in Issue #400 of *Circuit Cellar*.

WHAT IS SONATA?

At its heart, Sonata runs the CHERIoT Ibex core—a 32-bit RISC-V CPU core (Ibex) extended to support CHERI capabilities. Ibex is a small, efficient open-source core widely used in embedded designs, and the CHERIoT modifications add hardware support for CHERI's tagged pointers, bounds-checking, and compartmentalization features. The system is designed to feel like a "normal" microcontroller but with built-in enforcement of memory safety and isolation. The Sonata SoC (implemented on the Sonata board's FPGA) includes this CHERIoT-enabled CPU along with a suite of typical microcontroller peripherals and system components.

The Sonata board is shown in **Figure 1**. This looks like a normal sort of microcontroller development board, with the typical headers you might find on development boards. This board can fit a variety of off-the-shelf expansion boards including Arduino Shields, Raspberry Pi Hats, and others.

A block diagram of the board is shown in **Figure 2**, which illustrates the combination of FPGA-implemented microcontroller blocks along with the physical support circuitry on the board. Besides the Ibex core, which I already mentioned, it has a number of other peripherals worth discussing.

First, the system contains several types of memory. To start with, it includes 128KB on-chip SRAM with CHERI tag support (used for both instructions and data). Sonata also provides an external 8MB HyperRAM chip; the design allows executing code directly from HyperRAM with an instruction cache, which is useful for larger programs, though the external memory is slower. This HyperRAM can be thought of like flash memory in a typical microcontroller, larger but a little slower than accessing the SRAM.

Special features for the memory tag support are also needed. A hardware revoker unit is included to clear tags from memory as needed (for example, when memory is freed) to ensure deterministic use-after-free protection by preventing stale capabilities from being reused.

Besides these memories, more common microcontroller peripherals are also included. This includes multiple UARTs, SPI and I²C controllers, GPIOs, a timer, a PLIC interrupt controller, and even a USB device interface.

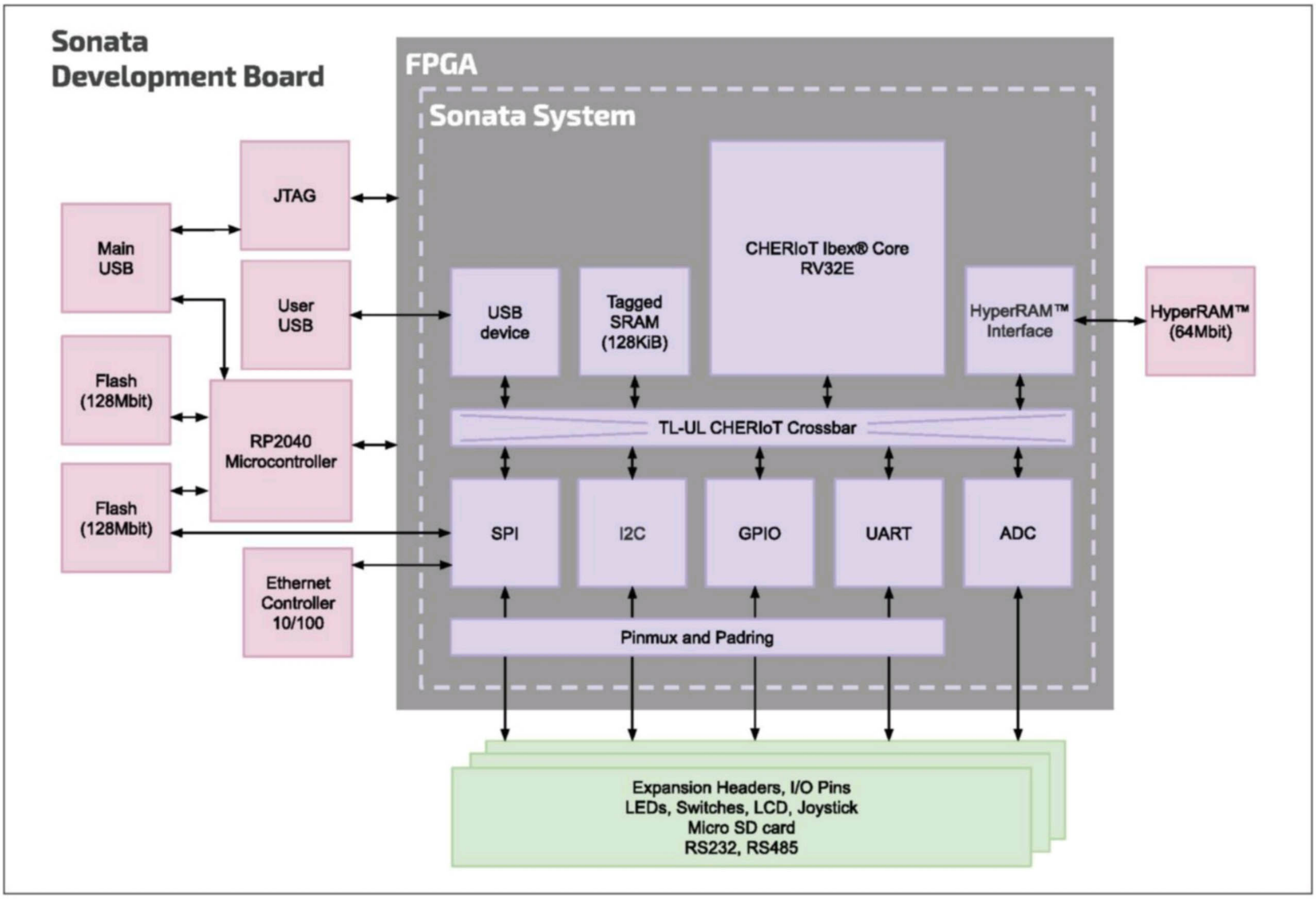

Sonata Development Board
FPGA
Sonata System
JTAG
Main USB
User USB
Flash (128Mbit)
RP2040 Microcontroller
Flash (128Mbit)
Ethernet Controller 10/100
USB device
Tagged SRAM (128KiB)
CHERIoT Ibex® Core RV32E
HyperRAM™ Interface
HyperRAM™ (64Mbit)
TL-UL CHERIoT Crossbar
SPI
I2C
GPIO
UART
ADC
Pinmux and Padring
Expansion Headers, I/O Pins
LEDs, Switches, LCD, Joystick
Micro SD card
RS232, RS485

FIGURE 2
This diagram shows the various blocks on the Sonata Board, some of which are implemented inside the FPGA in a soft-core microcontroller.

Some of these SPI controllers are dedicated to specific board functions such as the LCD and Ethernet, but the soft-core nature of the FPGA means you can always adjust that if you want!

There's also support for analog input via an ADC and PWM via dedicated controllers. All these peripherals are integrated into the CHERI-aware bus fabric, meaning their memory-mapped registers can be accessed through capabilities. In CHERIoT, device registers are typically exposed to software compartments via MMIO capabilities—a compartment's import table will include only the device regions it should access. This allows hardware access to be controlled at the compartment level. For example, only a trusted driver compartment holds the capability for a certain GPIO port. This sort of security compartmentalization allows you to perform fundamental security improvements such as ensuring that compromised code from one interface cannot access another interface. Unlike with a OS or RTOS, this is enforced in hardware, and the granularity is such that even an exploit into privileged (or secure mode) code cannot bypass these protections.

Compared to some of the existing security solutions that have only, for example, a secure/privileged and a non-secure/user mode, the higher level of granularity in CHERIoT makes it natural to design compartments around individual subsystems or peripherals.

To feel like a "real" microcontroller, the Sonata system provides typical pin multiplexor logic as well. This allows you to reroute peripherals between various pins. One unique feature is dedicated CHERI exception LEDs that illuminate when a security violation (like a capability fault) occurs. This provides immediate visual feedback if, say, a compartment tries to access memory out of bounds. These are great for teaching or educational environments, where you might be triggering those sorts of exceptions on purpose.

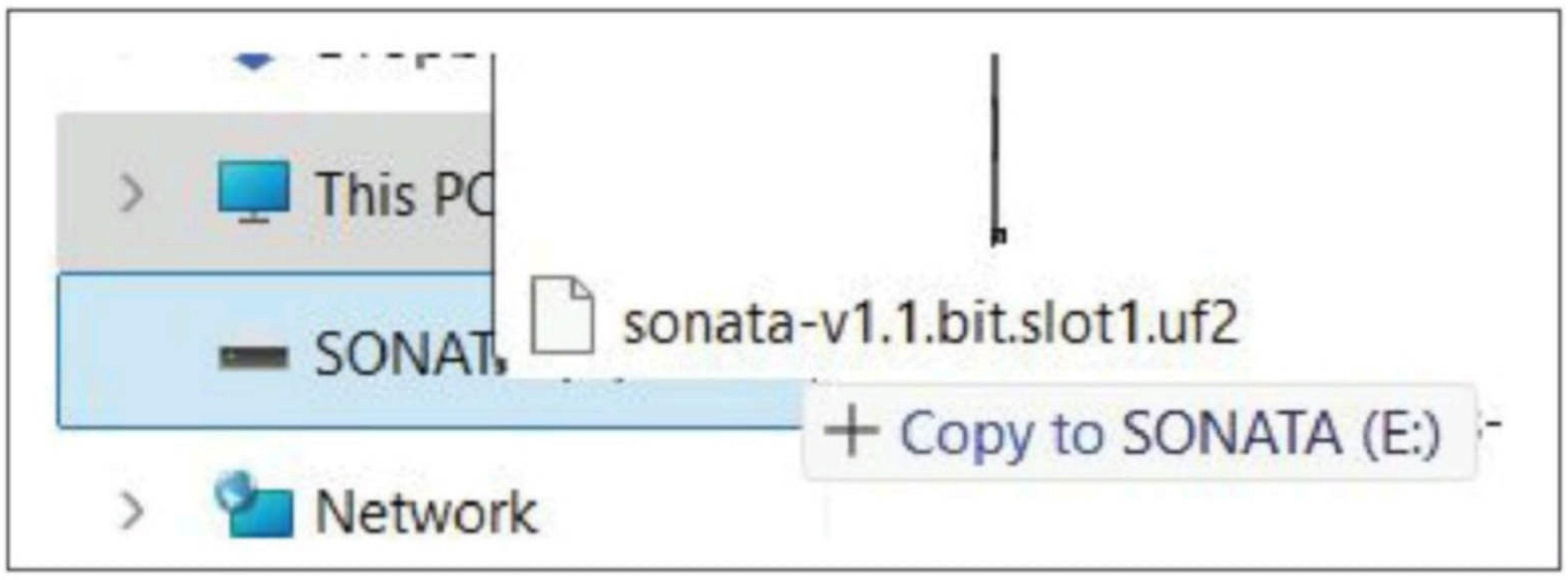


FIGURE 3
A simple drag-and-drop interface allows you to program the board from any computer.

BOOTING SONATA

With the board in front of you, what would be the next step? One difference from a normal development board is you can update the "microcontroller" itself, since that is just a FPGA bitstream.

You can download a "bitstream," which is the prebuilt microcontroller setup. One feature of the Sonata system is it can store up to three bitstreams onboard, which allow you to switch between different "cores" if you want. For now I'd download and install the 1.1 release of Sonata (the most recent at the time of writing this article).

This download takes the form of a .uf2 file, which is a generic file format image. Programming the board just requires me to drag it onto the USB drive which appears when plugging in the Sonata board, as in **Figure 3**. The "slot" number should match the switch position I plan on using. In the future I could have different bitstreams if I wanted, which can be helpful if for example you want one version with a lot of SPI peripherals.

This same drag-and-drop interface can also be used for programming software that we'll build. Under the hood this is handled by a RP2040 (Raspberry Pi Pico), which decodes the UF2 file format and writes it into SPI flash memories on the board. Of course, all of this firmware is open-source if you'd like to implement it on your own board.

If you want a little more control over programming (or debugging), the board also has a FTDI chip, which provides JTAG and serial interfaces. You can use the JTAG interface to debug both the FPGA and the microcontroller core, but that is beyond the scope of what I can cover in a short intro article.

TOOLCHAIN SETUP

To build for the Sonata board, you'll need a CHERIoT-enabled toolchain. This includes a compiler that understands the CHERIoT extensions, as well as the CHERIoT RTOS libraries. The two quick methods of getting this setup are to install them locally using Nix, or to use a VS Code "dev container" available with the CHERIoT-RTOS project.

I decided to proceed with the local install. I'm on Windows and following the "Getting Started" guide successfully had a toolchain running on WSL on Windows (see the link in additional materials). Be sure to carefully follow the requirements for Windows,

ABOUT THE AUTHOR

Colin O'Flynn (colin@oflynn.com) has been building and breaking electronic devices for many years. He is CTO of NewAE Technology based in Halifax, NS, Canada and is currently an assistant professor at Dalhousie University. Some of his work is posted on his website at www.colinoflynn.com.

including how to download Nix and additional WSL settings required. My first attempting of using a default Nix install, and not based on the link in the guide, caused myself problems!

From here, the process of checking out the required software and building all the examples is given in **Listing 1**. If this is successful you'll end up with a bunch of UF2 files in a "build" directory. Each of those files is one of the built examples.

You might notice the filenames have a suffix of "slot1", "slot2", or "slot3". For example, one of the LEDs and LCD demo filenames is `leds_and_lcd.slot2.uf2`.

This is because the board has several "slots" you can load the software into, just like with the bitstream. Currently there are three slots, you can select which one to boot from with the little selector switch (see **Figure 4**).

To program it I can just drag-and-drop the associated UF2 file onto the SONATA drive that we used for programming the bitstream. The UF2 file format specifies within the file whether it's a bitstream (for the FPGA) or software.

With examples built, we can now look at how you can use some of the features of CHERIoT.

HARDWARE ACCESS CONTROL

I'm going to follow the Hardware Access Control tutorial from the Sonata Software repository, again check the resources for the full link. This is a great tutorial as it demonstrates how you can easily add very fine-grained access control to your code with minimal effort.

Start by looking at the code in **Listing 2**, which is part of the C++ code for one thread in a simple "blinky" demo (which blinks some LEDs). On line 1 we are requesting the LED with index 7, so getting LED7 on the Sonata board. This LED is "acquired" on line 5 in the code with the `acquire_led()` call. So far this looks like a typical HAL (Hardware Abstraction Layer) provided by a board vendor.

The first obvious sign of something different is however on Line 3. You'll notice that a special compiler directive is being used, which defines a `__cheri_compartment()` name. These compartments can be used to provide granular access control. The rest of the code is normal C/C++, and you can see that it simple blinks the given LED on lines 9 through 13. Note that I've cut the headers out of this file to save space, you can see the full source code in the linked tutorial.

We need something to "conflict" with this thread to have any sort of interesting demos and understand what the code is doing. That is provided in Listing 3, which is another thread that is also blinking some LEDs. In this case it's trying to "walk" through the LEDs being blinked in sequence. The conflict happens because the `for()` lop on Line 15 will eventually try to toggle LED7, the exact same LED that we were toggling in Listing 2!

The code in Listing 2 and **Listing 3** have taken advantage of another CHERIoT feature;

```
git clone --branch v1.0 --recurse-submodule \
    https://github.com/lowRISC/sonata-software.git
cd sonata-software
nix develop
xmake -P examples
```

LISTING 1
To set up and build the software repository, these few lines should "just work".

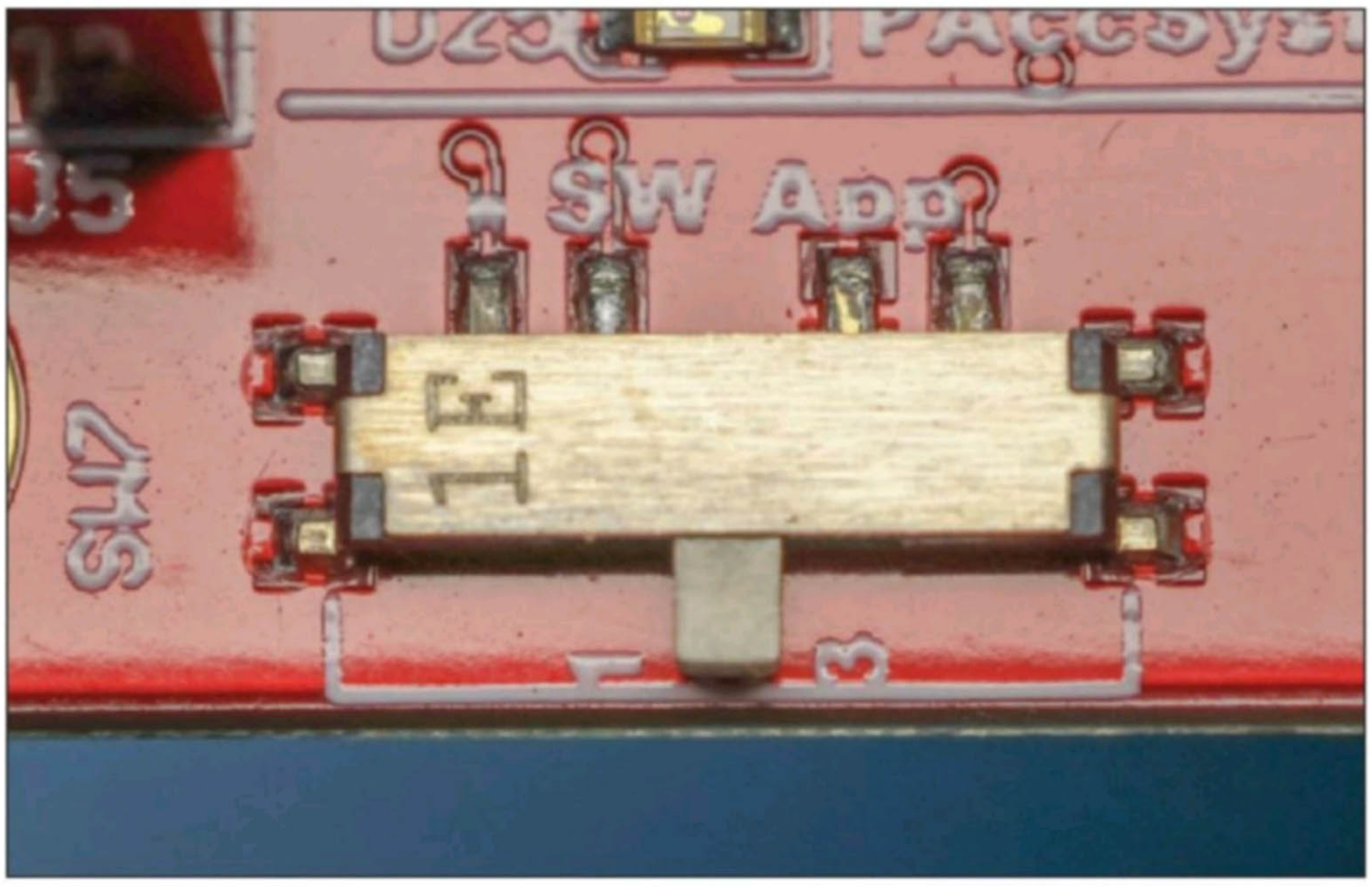


FIGURE 4
Here I can select various software binaries to run. Be sure it matches the slot number you programmed.

```
1 : static constexpr uint32_t LedIdx = 7;
2 :
3 : void __cheri_compartment(“blinky_dynamic”)
start_blinking()
4 : {
5 :   auto ledOpt = acquire_led(LedIdx);
6 :   Debug::Assert(ledOpt.has_value(), “LED {}
couldn’t be acquired”, LedIdx);
7 :   auto led = ledOpt.value();
8 :
9 :   while (true)
10:   {
11:     bool success = toggle_led(led);
12:     Debug::Assert(success, “Failed to toggle an
LED”);
13:     thread_millisecond_wait(500);
14:   }
15: }
```

LISTING 2
One thread in the blinky demo.

the call to `acquire_led()` from one thread will block the other thread from using it. In this case it’s a question of which thread was initialized first, that thread ends up acquiring access to the LED.

In this example, it’s actually Listing 3 that is called first, as its priority is set higher in the configuration. On Line 8 of Listing 3 we call `acquire_led()` in a loop, which acquires all the first eight LEDs. A later attempt to use the same LED from the other compartment is blocked by CHERI via the `acquire_led()` function call.

Of course, you might accuse me of trying to sell you old technology as something new: you could surely accomplish the same thing with an RTOS mutex (or lock), so why the fuss about CHERI? The reason is that a RTOS mutex is enforced only by the RTOS, a vulnerability may still allow an attacker to directly access the resource. An attacker who achieves a secure mode exploit is going to skip the call to `acquire_led()` and attempt to write directly to the GPIO peripheral memory space.

It’s this attack that can be blocked by CHERIoT. The memory space for the GPIO can be restricted to only be accessible through the `acquire_led()` interface. This is an incredibly powerful defense as it does not just rely only on a “secure” and “non-secure” mode as many current security protections assume. The problem with these is it can be difficult to guarantee all “secure” code is bug-free, as when you only have two levels of access an attacker who manages to find an exploit in the “secure mode” code can suddenly access all other systems. With CHERIoT we just need to make the individual drivers bug-free. Someone who wants to access a specific subsystem would need to find an exploit in the code within that compartment, not any other compartment.

```
1 : static constexpr uint32_t NumLeds = 8;
2 :
3 : void __cheri_compartment(“led_walk_dynamic”)
start_walking()
4 : {
5 :   std::vector<LedHandle *> leds;
6 :   for (uint8_t num = 0; num < NumLeds; ++num)
7 :   {
8 :     auto led = acquire_led(num);
9 :     Debug::Assert(led.has_value(), “LED {}
couldn’t be acquired”, num);
10:     leds.push_back(led.value());
11:   };
12:
13:   while (true)
14:   {
15:     for (auto led : leds)
16:     {
17:       bool success = toggle_led(led);
18:       Debug::Assert(success, “Failed to toggle an
LED”);
19:       thread_millisecond_wait(500);
20:     }
21:   }
22: }
```

LISTING 3
The second thread in the blinky demo.

Such exploits become challenging to avoid when you consider that many microcontrollers also involve vendor-provided libraries. It is common to have certain features, especially wireless stacks, to be provided by vendors in binary-only format. If those vendor libraries run in a privileged mode, an attacker who finds an exploit in a vendor stack can compromise any products using that stack. With CHERI's compartmentalization, such attacks are blocked.

If you’re interested in more details of how you could use CHERIoT, be sure to check out the Sonata Software documentation and exercises (linked in the additional materials). This is a great set of exercises and examples from the lowRISC team to show you how to use CHERIoT on the Sonata board, and includes various examples of using features like I^{2}C, the LCD, GPIO, and more.

CHERI FUTURES AHEAD

If this article interested you, you can see more information on the “Sonata System” documentation (see additional material). All the steps I’ve included in this article are from the “Sonata Software” getting started guide, also in the resources.

Besides these specific getting started guides, I also want to point to a number of resources available on CHERIoT.org, including the “CHERIoT Programmer’s Guide” by David Chisnall. It includes some topical posts about the CHERIoT security model, including a comparison of recent embedded security flaws that could have been prevented with CHERIoT.

If you’re interested in CHERIoT you might look at the CHERI Alliance, which is driving adoption of this technology across industry and academia. It’s exciting to see the many resources available for CHERIoT work, and even the state of all these examples from when my last article on the issue was published in November 2023. Hopefully you find this interesting, and even if you don’t run the examples yourself it’s a technology to keep an eye on for the future.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials

Resources

CHERI Alliance | https://cheri-alliance.org

Sonata | www.sunburst-project.org

JACOB BENINGO
CEO & Founder
EMBEDDED SOFTWARE ACADEMY BY BENINGO

Embedded Software Academy

Get Expert Guidance to Start, Grow and Modernize your Embedded Software Skills

Save Time **Save Money** **Advance Career**

Learn More: → **www.beningo.com**

The Premier Online Conference for Embedded Systems Engineers

Advance Your Skills. Accelerate Your Career.

- ✓ Expert-led training at an unbeatable value
- ✓ Learn on your schedule—unlimited on-demand access until Dec 2025
- ✓ Actionable insights you can apply immediately

Don't fall behind—stay ahead with the latest in embedded systems.

Embedded Online Conference

KEYNOTE SPEAKERS

Jacob Benningo

Elecia White

Philip Koopman

James Grenning

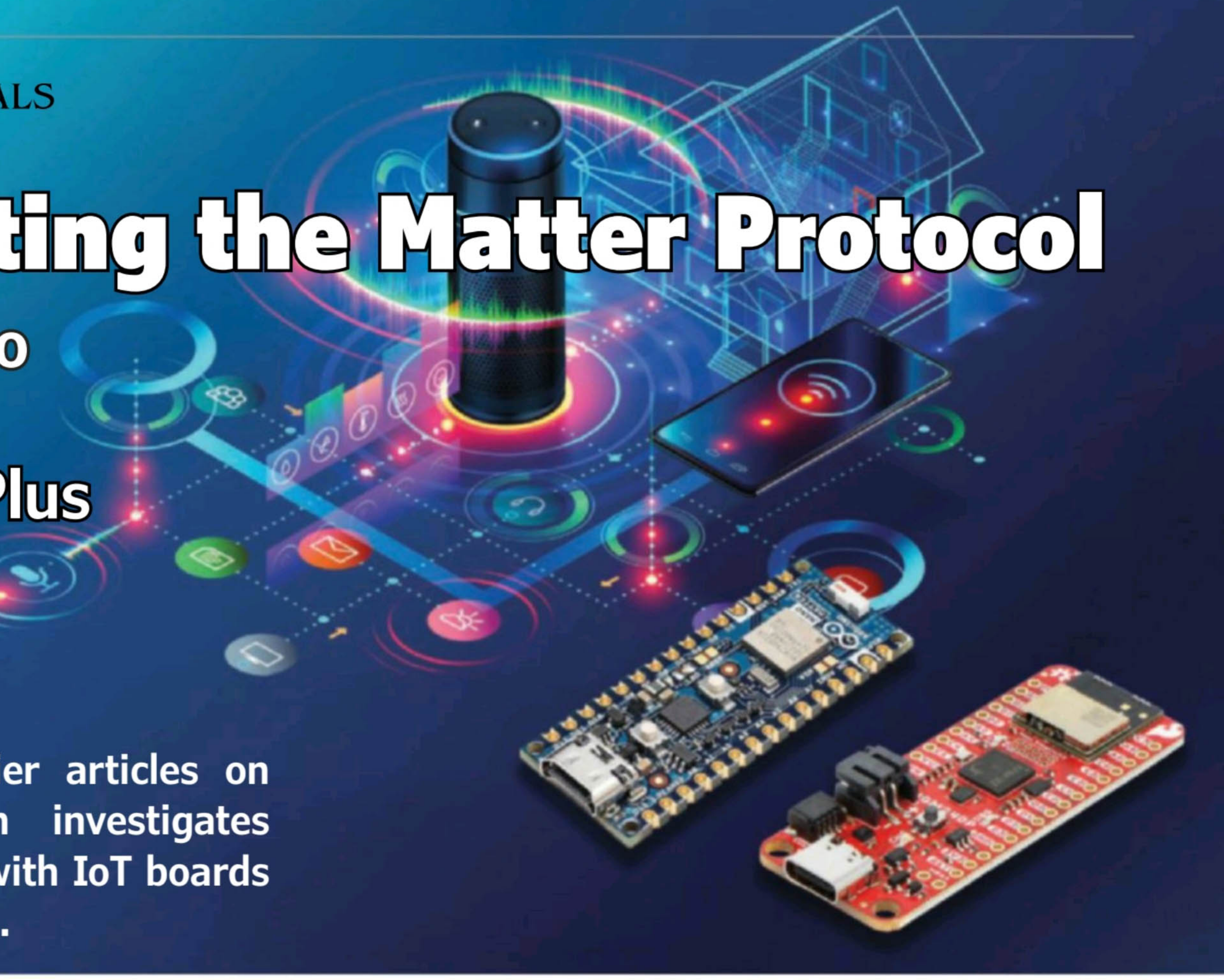
Register now and take your expertise to the next level!

Learn More: → **www.embeddedonlineconference.com**

PICKING UP MIXED SIGNALS

Implementing the Matter Protocol

Using the Arduino Nano Matter or Sparkfun Thing Plus Matter Boards



Following up on his earlier articles on Alexa connectivity, Brian investigates using the Matter protocol with IoT boards from Arduino and Sparkfun.

By Brian Millier

I've been building IoT projects lately that can be integrated with the Amazon Alexa Echo smart speakers in my home. I find the Alexa voice recognition capabilities to be excellent, and it's easy to set up routines for the various sensors and controlled devices that I've connected to it. Up until recently, the projects that I built for this purpose were only compatible with Alexa by virtue of their being an emulation of a commercial product (see my articles in issues 412 and 414).

However, when Arduino announced its Nano Matter development board, it promised to provide compatibility with my Alexa devices using a combination of the Thread and Matter protocols. This seemed to offer a higher level of integration than I had been able to achieve with my projects, which merely emulated some popular commercial devices (i.e., Philips Hue lightbulbs).

The Arduino Nano Matter board achieves Matter compatibility by using an RF-enabled MCU by Silicon Labs: the MGM240S. I use the Arduino IDE regularly and felt that having an Arduino-based Matter library available would simplify the programming process considerably. Before the Nano Matter board was available to customers, Arduino ported Silicon Lab's Matter library to the Arduino platform. I'm using Arduino IDE version 2.3.4 and downloaded version 1.0 of the board support package (BSP) for the Nano Matter board. I was able to compile several of their example sketches with no errors displayed.

However, when I received my Nano Matter board during Arduino's Early Adopters promotion, I was unable to get it to work with any of the examples, apart from the lightbulb example (connecting to my Alexa hub). However, a few months later, when BSP version 2.2.0 became available, I downloaded it and suddenly everything started working correctly with my Alexa devices.

In this article, I'm going to outline the features of the Arduino Nano Matter board. I'll also describe Sparkfun's Thing Plus Matter board, which is also based upon the same Silicon Labs MCU. Actually, the Sparkfun Thing Plus board became available somewhat earlier than the Arduino Nano Matter board—Sparkfun saw the advantages of the availability of the Arduino Silabs BSP, to target the large Arduino community. Let's start by looking closely at the Silicon Labs MGM240S RF-enabled MCU that is found on both boards.

THE SILICON LABS MGM240S RF-ENABLED MCU

Generally speaking, older Arduino development boards didn't contain either Wi-Fi or Bluetooth capability. Some newer models contained a separate RF module, handling one or more of these RF protocols, such as the u-blox NINA-W102. Also, there were a few Arduino boards, which contained an ESP32 for WiFi/Bluetooth or a Nordic nRF52840 (Bluetooth only).

The Nano Matter board contains an RF-enabled MCU: the Silicon Labs MGM240S. This device contains an ARM M33 processor as the main MCU, as well as an ARM M0 co-processor acting as the Radio Controller. This MCU handles Bluetooth LE as well as the 802.15.4 protocol, which is the basis for Zigbee, Open Thread, and the Matter protocols. In the case of the MGM240S, all these RF protocols operate on the 2.4GHz band. Note that this MCU does not handle the Wi-Fi communication protocol.

Note that the MGM240S MCU is part of Silicon Labs' EFR32MG24 wireless SoC family, which is a bit confusing, considering the variation in naming (MG24 vs. MGM24). If you are doing any in-depth work with this MCU, you'll need to download the EFR32XG24 Wireless SoC Reference manual from this URL: https://www.silabs.com/documents/public/reference-manuals/efr32xg24-rm.pdf [1].

The MGM240s has a maximum transmit power of +10dBm, which is 10dBm lower than the typical ESP32 transmit power level for Wi-Fi operations, but comparable to the ESP32's transmit power for the Bluetooth mode. The MGM240S has a very sensitive RF receiver at -104.5dBm, (at 250Kbps data rate). I'll discuss power consumption in more detail later. However, where you would normally expect an ESP32 to draw an average of 50mA to 70mA with lots of sharp, 350mA current spikes during RF operations, my measurements on the Nano Matter board showed a steady 5.7mA current draw when running a Matter sketch (although not using the maximum available Tx power). Similarly, the MGM240S has much lower sleep current in the various sleep modes, than the ESP32. I haven't been able to get an ESP32 to operate properly below 3V, but the MGM240S is rated down to 1.8V. All in all, the Nano Matter board is a good candidate for battery-operated devices. Like most ARM MCUs, the MGM240S runs on 3.3V and is not 5V-tolerent in terms of either the power supply or the IO lines.

The MGM240S has a very nice complement of I/O functions. It has the usual asynchronous serial ports- a single USART and two extended EUSARTs. Even its standard USART can handle SPI, Smartcard (ISO7816), IrDA and I^{2}S. The two EUSARTs add DALI (Digital Addressable Lighting Interface) to the list of supported protocols. While you can configure the three (E)USARTs for these specialized protocols, the current Arduino board support package only includes SPI and UART support on these three ports.

The MGM240S contains two I^{2}C ports. While I have used the I^{2}C port on the Nano Matter board, I have not tried the setClock() function to see whether it can handle higher bus speeds, like 400kHz and up, but that function is part of the Wire library.

This MCU contains 32 GPIO lines and the Nano Matter board has pinouts for 22 of these lines. The various internal peripheral functions (USART, I^{2}C, Timers, ADC, etc.) can be routed to various GPIO lines using the ABUS and DBUS multiplexers. You don't have to worry about doing this routing yourself, as the Arduino BSP has assigned the various peripherals to default pins. These are shown in the Pinout diagram found in the Nano Matter User Manual, or by referring to **Table 1**.

Note the pins labeled ADC. The MGM240S has a 12-bit ADC, which is a higher resolution than you would normally expect with MCU-based ADCs. Not only is this ADC a higher resolution, but it is also quite fast at 1Msps. The ADC can be configured for oversampling, which can extend the resolution to 16-bits at a lower rate of 76.9kbps. I doubt that you can do any such re-configuration using the Arduino analogRead() commands- apart from the ability to change the ADC reference from the internal 1.21V band-gap reference, VDD, 0.8VDD or an external 1.25V reference (connected to the AREF pin). The ADC is multiplexed to many of the MGM240S' GPIO lines, and 8 of these are pinned out on the Nano Matter board. The ADC is preceded by a Programmable Gain Amplifier with gains from 0.5x to 4x, but this feature is not implemented in the Arduino BSP.

MGM240S	Arduino Port name	Usage options
PB00	A0 / DAC0	GPIO / ADC / DAC
PB02	A1 / DAC2	GPIO / ADC / DAC
PB05	A2	GPIO / ADC
PC00	A3	GPIO / ADC
PA06	A4 / SDA	I^{2}C / GPIO / ADC
PA07	A5 / SCL	I^{2}C / GPIO / ADC
PB01	A6 / DAC1	GPIO / ADC / DAC
PB03	A7 / DAC3	GPIO / ADC / DAC
PB04	D13 / SCK	SPI / GPIO / ADC
PA08	D12 / MISO	SPI / GPIO / ADC
PA09	D11 / MOSI	SPI / GPIO / ADC
PD05	D10 / SS	SPI / GPIO
PD04	D9	GPIO
PD03	D8	GPIO / ADC
PD02	D7	GPIO / ADC
PC09	D6	GPIO / ADC
PC08	D5 / SCL1	I^{2}C / GPIO / ADC
PC07	D4 / SDA1	I^{2}C / GPIO / ADC
PC06	D3 / SS1	GPIO / SPI / ADC
PA03	D2 / SCK1	SPI / GPIO
PA05	D1 / PIN_SERIAL_RX1 / MISO1	UART / SPI / GPIO / ADC
PA04	D0 / PIN_SERIAL_TX1 / MOSI1	UART / SPI / GPIO / ADC

TABLE 1
This shows the relationship between the Arduino port names and the corresponding MGM240S GPIO pins. It also shows the default peripheral assignments for each of the pins.

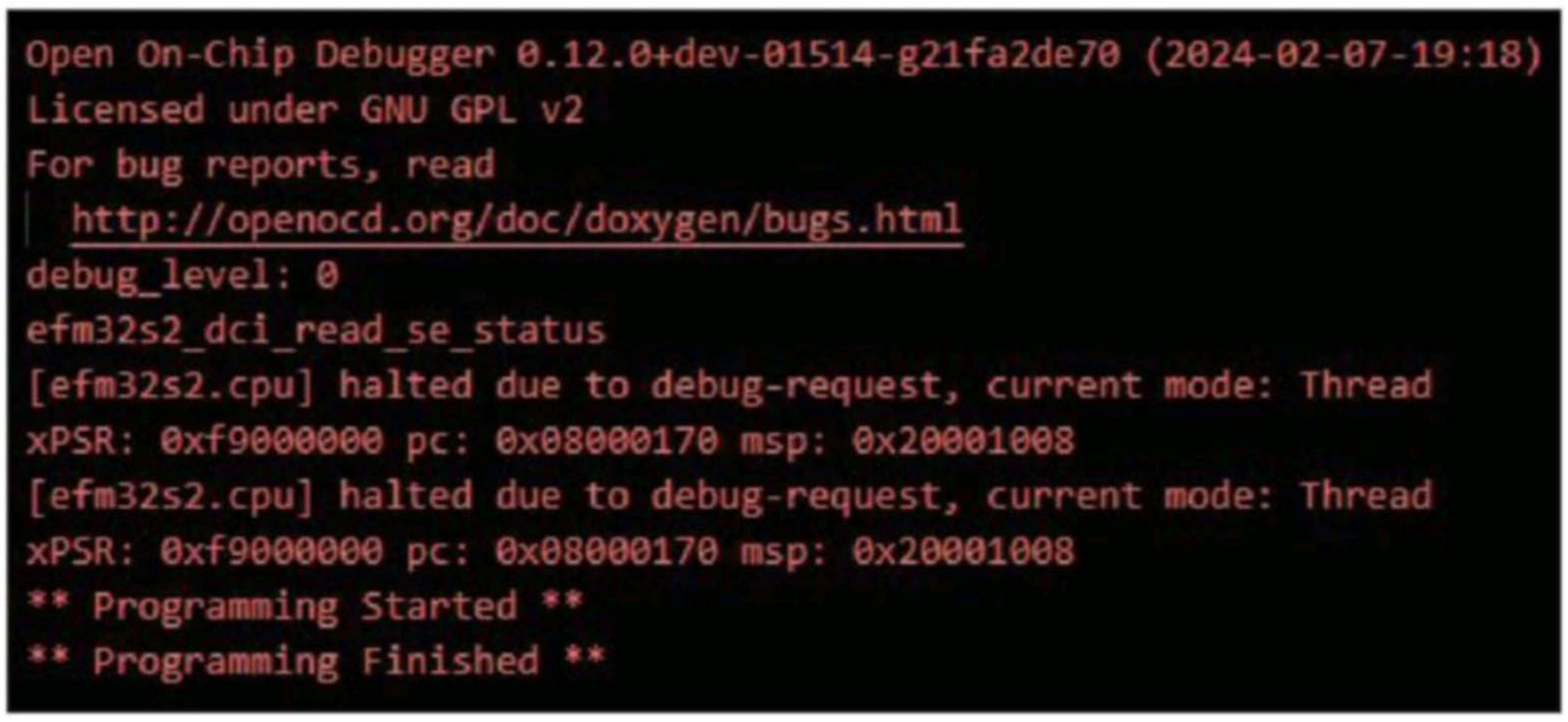
As an extra bonus, the MGM240S contains two 12-bit DACs—this is a feature that most MCUs don't have. The DACs are followed by rail-to-rail buffers and can be combined to produce a single differential output, if desired. While these DACs can be fed by the MCU's LDMA controller, when using the Arduino BSP, the LDMA controller is not available and the AnalogWrite() command is what is used.

Rounding out the analog peripherals, there are two analog comparators, which are not supported by the Arduino BSP. There is also an MCU die temperature sensor which is available using the BSP's getCPUTemp() function.

Changing the pins for the various peripheral devices from their default assignment can be done by manipulating MCU registers. I didn't attempt this—the Reference manual is 1100+ pages and there are many hundreds of registers!

The MGM240S' GPIO lines can be configured as outputs or inputs (with various pullup, pulldown options). There is nothing unusual here except that the GPIO outputs can source or sink 20mA, which is higher than you normally see on ARM MCUs. Those 20mA figures are what is quoted in the Silicon Labs MG240S datasheet. The Arduino documentation for the Nano Matter board quotes 40mA Source and 28mA sink. The Arduino figures may apply to a single GPIO line being driven, while the Silicon Labs' numbers may be a more conservative, general usage figure.

The MGM240S contains a 16-channel Peripheral Reflex System (PRS). On first glance, you might think that this is the GPIO pin multiplexer function, but it's not. Instead, it is a way to route various peripheral device inputs/outputs to other peripherals without involving the MCU itself. A routine use of this would be to configure an MGM240S timer to trigger an ADC or DAC at a fixed rate. The Arduino BSP does not currently support the PRS subsystem.



```
Open On-Chip Debugger 0.12.0+dev-01514-g21fa2de70 (2024-02-07-19:18)
Licensed under GNU GPL v2
For bug reports, read
  http://openocd.org/doc/doxygen/bugs.html
debug_level: 0
efm32s2_dci_read_se_status
[efm32s2.cpu] halted due to debug-request, current mode: Thread
xPSR: 0xf9000000 pc: 0x08000170 msp: 0x20001008
[efm32s2.cpu] halted due to debug-request, current mode: Thread
xPSR: 0xf9000000 pc: 0x08000170 msp: 0x20001008
** Programming Started **
** Programming Finished **
```

FIGURE 1
This is the status message displayed after you have downloaded a sketch to the Nano Matter board.

The MGM240S MCU contains many timers, several of which operate in various low power/sleep modes. Not many of these timers are directly available via the library functions contained in the Arduino BSP. However, the standard Arduino millis() and micros() functions are available. Also, if you use the deep sleep mode, you can configure it to wake up via a GPIO pin change or a millisecond timer.

In my projects, I seldom get involved with any security measures that may or may not be present in an MCU. For interest's sake I'll mention that the MGM240S contains the following security features:

- Secure Boot with Root of Trust and Secure Loader
- Hardware Cryptographic Acceleration
- True Random Number Generation
- ARM Trustzone
- Secure Debug interface lock/unlock
- Secure Key management and Anti-Tamper provisions

UPLOADING CODE AND DEBUGGING

Getting your code into the MGM240S is different than what you may be accustomed to, if you routinely use other ARM MCUs or the ESP32. The ESP32 contains its own bootloader in ROM and uploads code via its serial port, using a Serial-USB bridge chip. Newer ESP32 MCUs can upload code directly using their onboard USB port. Many ARM chips (RP2040, etc.) come with a UF2 ROM bootloader, which causes the MCU to appear as a flash drive to the host computer, allowing drag/drop programming.

The MGM240S MCU is programmed via its software debugging port (SWD). This would normally mean that you need a programming/debugging device, such as those made by Segger, J-Link and others to go between the MGM240S and your host PC.

The Nano Matter board contains an Atmel SAM-D11-D14A Arm MCU, which has been programmed to perform the following functions:

- Upload your code to the MGM240S via a USB port.
- Perform hardware debugging
- Implement a serial-USB bridge function-useful for software debugging using Serial.println() statements.

ABOUT THE AUTHOR

Brian Millier runs Computer Interface Consultants. He was an instrumentation engineer in the Department of Chemistry at Dalhousie University (Halifax, NS, Canada) for 30 years.

Generally speaking, you would need to load a driver for debugging devices such as the Segger units, as well as installing a debugging application like OCD. However, when you load the Arduino "Silicon Labs" board support package, this is taken care of for you. If you are not going to be doing any hardware debugging, all that is necessary to upload code to the MGM240S is to press the normal upload icon present in the Arduino IDE. While the uploading is taking place, you will see a status report detailing what is happening, and it's obvious that the OCD software on the PC is handling the uploading process. **Figure 1** shows the status screen after a successful program upload.

The Sparkfun Thing Plus Matter board works in much the same manner but instead of using the SAM-D11 MCU for the debug interface, it has a second Silicon Labs MCU (EFM32GG128416F1) onboard. In addition to this debugger MCU, they also needed to add two EEPROMs: one for the debugger ID and the other for Serial Flash memory.

While I have had some experience with hardware debugging on other MCUs that I have used in the past, in those cases hardware debug support was built into that MCU's development IDE. Also, the actual debug hardware was contained on the development board itself, and the IDE would automatically confirm the existence of the debugger as well as identify the specific target MCU.

The original Arduino IDE was never designed for hardware debugging, since the 8-bit Atmel Mega328 AVR MCU didn't support it. When Arduino expanded its focus to handle 32-bit ARM MCUs, hardware debugging was added to the IDE, but it was never really integrated into the IDE, nor was it easy to use. I haven't spent enough time investigating debugging with the Nano Matter board to include any details in this article.

If you are using the various MCU hardware peripheral functions supported by the Arduino BSP, you may be interested in the folder location containing the related source code. These are "buried" at the following, rather deep file hierarchy:

Windows C > Users > Username > AppData >
Local > Arduino15 > Packages >
SiliconLabs > Hardware > Silabs > 2.2.0 >

Below that level, there are folders for "cores" and "libraries," which contain the source code for various hardware (peripheral) functions and other standard functions such as the Matter routines.

A COMPARISON BETWEEN THE TWO BOARDS

In terms of implementing the Matter or BLE protocols, the two boards are equally capable, since they both use the same MGM240S MCU. However, each board has its own distinct features. I've highlighted some of those differences in **Table 2**. If compact size is important, the Arduino Nano Matter is clearly the smaller of the two boards. If you are designing a carrier PCB, each of the boards uses a different, but industry-standard, footprint. The Sparkfun board is well-equipped for operation with a LiPo cell, including a JST socket, charger and a dedicated battery fuel gauge chip. Both boards contain 21 or 22 GPIO pins but the Sparkfun board also contains a QWIIC socket and a µSD card socket. I have connected a µSD card to the Nano Matter board, using the default SPI pins shown in Table 1, and it works properly. If you use the Thing Plus Matter's µSD card socket,

Feature	Arduino Nano Matter	Sparkfun Thing Plus Matter
Size	18 x 45 (smaller)	22.9 x 58.4 (larger)
Price	25 USD	25 USD
GPIO pins available	22	21
Board footprint	Arduino Nano	Feather-compatible
On-board LDO	Yes	Yes
LiPo support	No	JST socket, charger, fuel gauge
USB socket	USB-C	USB-C
QWIIC connector (I^{2}C)	No	Yes
µSD card socket	No	Yes
Pushbuttons	Reset, User	Reset
Built-in Antenna	Yes	Yes
LEDs	Power, RGB	Power, Charge, Status
Hardware SWD socket	No	10-pin Mini Simplicity
Maximum Input Voltage	22V max.	6V max.

TABLE 2
This is a comparison of the features of the Arduino Nano Matter and Sparkfun Thing Plus Matter boards.

it's also connected to the MGM240S using the default SPI port (MCU pins PC02, PC03, PC06 and PA07). Plugging a µSD card into the Sparkfun Thing Plus Matter board worked fine, using the SD card examples included with this board's BSP. In both cases, you must properly define which GPIO pin you are using for the SPI-CS line, in your sketch. The three GPIO lines driving the Nano Matter board's RGB LED (PC01, PC02, PC03) are not shared with the GPIO lines that are pinned out to the board's header connectors.

The Sparkfun Thing Plus Matter board contains a standard 0.05" 10-pin debug connector, labelled Mini Simplicity. This socket is connected directly to the MGM240S SWD and JTAG/trace pins. This allows you to use an external Segger or J-Link debugging tool instead of the on-board debugging circuit, implemented by the EFM32GG128416F1 MCU on the Thing Plus Matter board itself. To do so you would need to cut the link labelled LP on the bottom of the board. This places the EFM32GG128416F1 J-Link debugger in a low power sleep state (which tri-states its debugging pins).

If you are interested in the lowest possible current consumption, in deep-sleep mode, the Nano Matter board accommodates this in the following ways:

- The power LED can be de-activated by cutting a link on the top of the board.
- The SAM-D11 debugger is powered only when the Nano Matter board is powered via the USB port.
- In deep-sleep mode (with debugger and power LED de-activated), I measured the Nano Matter's current consumption at 1.5µA, with my Joulescope JS220 Energy Analyzer. This matches almost exactly the figure quoted in the MGM240S datasheet.

In the case of the Sparkfun Thing Plus Matter board, it's a bit more involved. You can deactivate the debugger, the Power and Charge LEDs, by cutting links on the bottom of the board. **Table 3** shows the current consumptions with various combinations of debugger/LED s disconnected. The first 5 entries were measured running a sketch that tested the µSD card (SD->ReadWrite sketch in Examples for Sparkfun Thing Plus Matter). The last 3 entries were measured while running a Matter sketch, with the last entry in a deep-sleep mode.

From Table 3, and my measurement of 1.5µA deep-sleep current for the Nano Matter board, you can see that the Nano Matter board is much better suited to very low-power operation- assuming that you place the MGM240S in a sleep mode for most of the time.

I should mention that you should be careful when trying out the deep-sleep function. If you write a simple test sketch that puts the MGM240S into deep-sleep at any time soon after the device comes out of reset, you won't be able to easily re-flash it with a new sketch. I found that the upload process would fail—presumably because the MGM240S has gone into a deep-sleep mode so soon after reset that the debugger chip can't perform the flash operation. If your intent is to have the MCU come out of reset for just a short time interval, before it goes into deep-sleep, I recommend that you add a bit of code, ahead of the deep-sleep command, allowing you to skip the deep-sleep, if a specific GPIO line has been shorted to ground. Then, you can just insert a jumper when you want to re-flash the MCU for any reason.

The Arduino Nano Matter uses the standard Arduino GPIO pin convention, A0-A8, D0-D13. The Arduino Silicon Labs BSP contains definitions to convert these Arduino pin definitions to the actual MGM240S GPIO pins. However, if you have a sketch that uses the actual MGM240S GPIO pin names, check out Table 1, which shows the relationship between the Arduino pin naming conventions and the MGM240S GPIO pin designations. Also, this table shows the different Peripheral functions and their default pins (for the Nano Matter board).

The Sparkfun Thing Plus Matter board uses a pin naming convention closer to the actual MGM240S GPIO pin designations. That is, the MGM240S GPIO pin PD02 is labelled D2 on the board. However, note that it must be referred to as PD2 (not PD02 or D2) in your sketch.

With the Sparkfun Thing Plus Matter board, I initially found that it looked like I could easily upload sketches, but they didn't work—at least in so far as they did not send out any startup message to the Serial port. Even the simple "ASCII table" example, which tests the Serial port, showed no response from the Serial port. At this point, I had not yet selected a programmer, but the status screen appeared to show that the uploads had succeeded. For no particular reason, I next decided to select Burn Bootloader from the Tools menu: this immediately prompted me to select a programmer. Once I selected the OCD programmer, the bootloader was installed properly. After this, any sketches I uploaded worked properly. I assume that the bootloader is not pre-installed by Sparkfun, and that it contains code needed to operate the Serial port, but this is just a guess on my part.

An issue I ran into with the Arduino Silicon Labs BSP concerned the SPI port library. The Sparkfun board has a µSD card socket on it. I tested it using the Arduino Silicon Labs BSP SD card example programs and it worked fine. I later added

Sparkfun Thing Plus Matter-Operating Condition	Current consumption
1) Debugger on, LEDs on, µSD card inserted, running SD card sketch	45.4mA
2) 1 above less µSD card inserted	24mA
3) 2 above less power LED	22.6mA
4) 3 above less Debugger chip	1.83mA
5) 4 above less charge LED	1.80mA
6) debugger on, all LEDs on, running Matter sketch	25.25mA
7) debugger off, all LEDs off, running Matter sketch	6.37mA
8) debugger off, all LEDs off, deep-sleep	337µA

TABLE 3
This table shows the current consumption of the Sparkfun Thing Plus Matter board under various operating conditions, including deep sleep.

a DF Robot DFR0995 SPI-based TFT display, which worked OK (using Adafruit's driver). However, when I later added the Matter protocol to my sketch, the TFT display stopped working. When I examined the SPI port signals on the 'scope, I found that the SPI port was operating at 40MHz. At this high clock rate, the SCLK signal was terrible—as shown in **Figure 2**. This screen capture was taken while the DFR-0995 display was working, but you can see that the first clock pulse is so low in amplitude that it's surprising that the display accepts it. When the µSD card was inserted, this clock signal degraded just a bit more, but the TFT display still worked. For some strange reason, adding the Matter protocol to the sketch also degraded the SCLK signal a bit, but not enough to stop the display from working. However, the combination of the Matter protocol plus having the µSD card installed, was enough to kill the DFR-0995 display.

It traced this problem to the "Adafruit_ST7789" library. This library configures the SPI port with a default clock rate of 32MHz. However, when running on an MGM240S MCU, this results in an actual SPI clock rate of half the system clock speed or about 39MHz. While the display worked at this clock rate initially, it was not reliable. Over time, the display would display lots of random characters and even switch background colors. In the "Adafruit_ST77xx.cpp" file, on line 38, I made the following change:

```
#define SPI_DEFAULT_FREQ 8000000 ///< Default SPI data clock frequency
```

The MGM240S MCU is able to generate this frequency exactly, and the much slower SPI clock rate eliminated all the display problems I had experienced before The input voltage range on the Nano Matter board is 4V to 22V, using a MC2322 switching regulator. The Sparkfun Thing Plus Matter board has a conventional LDO (XC6222B331), which has a much smaller maximum input voltage—6V.

BUILDING YOUR FIRST MATTER DEVICE

I bought my Arduino Nano Matter board during Arduino's Early Adopter promotion. At the time, Arduino's online documentation for the board was quite detailed, having separate sections describing how to connect the board to Google Home, Amazon Alexa devices, and more. This process is called commissioning. When this Early Adopter promotion ended, this detailed online documentation disappeared and was replaced with different content and tutorials, which cover only Google Home connectivity. This is unfortunate—had I only discovered the Nano Matter board after the Early Adopter promotion had ended, I would not have had access to the more detailed documentation, covering the Alexa hub that I have. I doubt I would have bought the Nano Matter board without having seen this earlier documentation.

What I describe in this section was tested using the Alexa smart home hub that I have in my home. The Early Adopter documentation stated that only certain Alexa devices would work- those that contained a Matter + Thread border router. I have an older model Echo device (4th Gen). This is the more expensive version of the Echo smart speaker lineup. I bought it for its superior sound performance, but also because it contains the Matter + Thread border router. Check with Amazon to see if your Alexa device has Matter + Thread capability.

Having mentioned Arduino's warning about Matter + Thread border router capability, I should add that when you commission the Nano Matter board using your Alexa device, you end up adding two devices: a Thread hub plus whatever Matter device you have programmed the Nano Matter board to emulate. I'm no expert on the Matter/Thread protocols, and I guess you need both a Thread border router and a separate Thread hub, which I found to be somewhat confusing.

When you load Arduino's Silicon Labs BSP, it will also install many example sketches. These cover some basic functions that you might do with any non RF-enabled board, but also include numerous examples of various Matter devices, as shown in **Table 4**. The entries with a ✓ are ones that I have tested out personally. Let's go through the commissioning process step by step.

MATTER COMMISSIONING

Before loading any examples, select your specific board and go to Tools, Protocol Stack and select "Matter". After loading one of the Matter examples, and running it, you

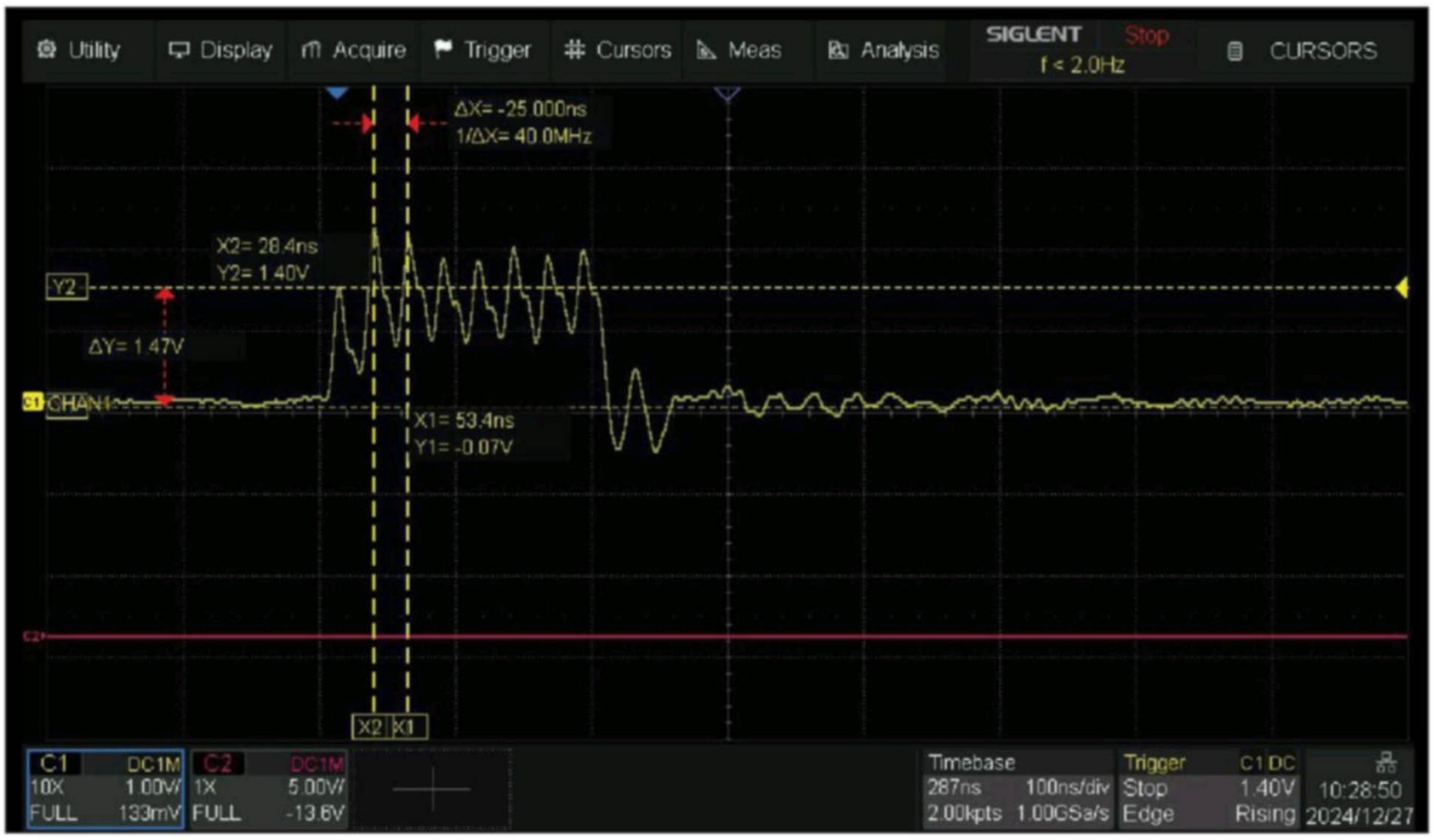


FIGURE 2
This is a scope capture of the Sparkfun Thing Plus Matter board's SPI SCLK signal. It is a marginal signal since it's using a very high 40MHz clock.

Air Quality ✓	Contact Sensor	Door Lock
Fan	Flow Sensor	Humidity Sensor ✓
Illuminance Sensor	Lightbulb (various)	Occupancy ✓
On-Off Outlet ✓	Pressure Sensor	Sensor and Bulb
Sensor (Multiple)	Switch ✓	Temperature Sensor ✓
Thermostat	Window Covering	Nano lightbulb-color ✓

TABLE 4
These are the Matter example programs supplied as part of the Arduino Silicon Labs Board support package. I have checked off the ones that I have successfully tested out.

Please scan with your CHIPTool app.

Payload: MT:6FCJ142C00KA0648G00

FIGURE 3
This is the QR code provided by the URL that is supplied by the Arduino Matter example sketches. The statement accompanying this code, saying that it is unique to this device, is not true.

will see a message on your Serial Terminal stating that the Nano Matter board is not yet commissioned. It will print out an 11-digit manual pairing number- which will be 34970112332. It will also display a URL, which points to a QR code which can be used instead of the manual pairing code (see **Figure 3**). The QR code would be the easiest method, assuming you are using the Alexa app on your cellphone.

Using the Alexa app, go to Devices and click on the "+" icon at the top right side of the screen. Then select "Add device". You then pick the device that best describes the example you have programmed into the Nano Matter board. Note that you will be giving the device a custom name later on, so the device type that you select here does not have to be a perfect match but should match in terms of whether it is a sensor (input) or a controller (output).

Once you've chosen the device, you'll be presented with a list of commercial brand names, but scroll through them until you see "other" and pick that. Then select the option labelled "Matter". You'll be asked if your device has a Matter logo, to which you will select "Yes". Also answer Yes when asked if the device is on. Next, either select "Scan QR code", which will launch your phone/tablet's camera app or select "Try numeric code instead" and type in the manual pairing code. The Alexa app will display "looking for your device" and should shortly display "This device isn't Matter certified". This message is legit- the Nano Matter board isn't certified for any device other than the lightbulb example. However, you must select Yes, to proceed. The app will display "connecting to your device". As this process is happening, your Arduino Serial Terminal should display:

```
Waiting for Thread network...
Connected to Thread network
Waiting for Matter device discovery...
Matter device is now online.
```

The Alexa app will display "Silicon Labs Hub and one component device found and connected. Pressing Next will advance to the settings page for both of those devices. The Hub will be labeled "First device" or "Second", etc. for subsequent boards (or other device examples that you try). Select this entry, press Next, and give this device the name "Nano Hub" or "Sparkfun hub"—whatever is meaningful to you. Then select the next entry which has been labelled "First motion sensor"—assuming you had selected a motion sensor device, earlier in the procedure. You can re-name that as you see fit.

At this point you can assign the device to a room and/or set up a routine for it—this is no different than what you would do with a commercial smart home module.

There are a few things I'd like to point out at this stage. I started out with a single Arduino Nano Matter board. When I navigated to the URL that provides the QR code, it states "This QR code is unique for your device". Knowing that the MGM240S MCU contains a unique ID number stored in its ROM, I assumed that both the manual pairing code and the QR code, were generated using this ID # in some way, and were indeed unique to the Nano Matter board that I was using. That is not the case: when I later got the Sparkfun Thing Plus Matter board, and followed the same procedure, it also gave the same manual pairing and QR codes. From this, I assume that both the pairing code and the QR code are something like the VID number of a USB device. It just represents that the product comes from a specific vendor and doesn't differentiate it any more than that.

I assume that the 34970112332 manual pairing code and associated QR code, have been assigned to either Silicon Labs or Arduino in terms of the Matter protocol. In other words, you can't commission your board using Alexa with just any 11-digit pairing code that you randomly choose.

After you have commissioned your Nano Matter board with Alexa, the Matter authentication information, supplied by the Alexa hub, is stored in the MGM240S' flash memory. Exactly where in flash is not documented, but it does **not** get erased when you upload either a modified version of your sketch or a completely new sketch, to the MGM240S.

What if you decide to try out/upload a different Matter example sketch to your Nano board? The first thing you will notice is that the Serial terminal will no longer display the message:

```
Matter device is not commissioned
Commission it to your Matter hub with the
   manual pairing code or QR code
Manual pairing code: 34970112332
```

The reason for this is that the sketch first checks flash memory for authentication information. However, this time it will find it. Therefore, it won't enter the routine in which it attempts to connect with the Alexa app and identifies itself as a new Matter device. Therefore, when you try to "add a new device", using the Alexa app, it will fail to find anything new. I also found that making even minor changes to a Matter-enabled sketch (not changing the device's Matter definition in any way) also required that the de-commissioning process be performed. Otherwise, my Alexa device wouldn't re-discover the presence of the device. The process described above is handled in the sketch by the following function `Matter.isDeviceCommissioned()`.

The de-commissioning documentation, that was available during the Arduino Nano Matter Early Adopter promotion, has disappeared. I don't see that information in the current Arduino documentation anywhere. However, if you wish to re-purpose your Nano Matter board to be a different sort of Matter device, you must first de-commission it, as mentioned above. There are two ways to do this:

- Load the example sketch "matter_decommission", which can be found among the other Matter examples supplied with the Arduino Silicon Labs BSP.
- In the Arduino IDE Tools menu, select the "Burn bootloader" option (the easiest way).

As well as this de-commissioning, if you re-program your Nano Matter to be a new device, you should check in the Alexa app to see if it still shows the original Nano Matter device that you added. If so, you should delete the original hub entry, which should also remove the original device as well. I note that the Alexa app doesn't always refresh discovered devices immediately- existing devices can take a while to show up and can disappear at times.

BLE CONNECTIVITY

The MGM240S MCU also contains BLE functionality. This is used during the Matter commissioning procedure, to allow a new device to advertise itself to the Matter host. When you specify the Matter protocol stack, it includes enough BLE functionality to allow this commissioning to work.

However, if you intend to run a sketch that includes standard BLE functions, you must specify the BLE stack in place of the Matter stack. From this I assume that you cannot have both Matter and BLE connectivity in any given Arduino sketch. I have not worked with Silicon Labs' development IDE: Simplicity Studio, at least in terms of MGM240S program development. Possibly, the Silicon Labs board support libraries can run both protocols in the same program (not necessarily at the same time).

The Arduino Silicon Labs BSP contains two BLE stacks- a Silicon Labs BLE stack and an Arduino BLE stack. The Arduino EZ-BLE examples require the Arduino BLE stack. For the Silicon Labs stack, there are some BLE examples under the heading "Silicon Labs" in the Examples section. While both BLE stacks perform much the same operations, the way the functions are written differ between the two stacks.

My only experience with BLE involves the NUS profile. This is the Nordic Uart Service: basically, a wireless bidirectional UART link, between a BLE client and BLE server. I use this to communicate between an RF-enabled MCU and an Android cellphone/tablet. I make use of the Android GUI-O app on the cellphone to implement a dashboard on the cellphone which is controlled by the external MCU. In the past, I've used ESP32 MCUs for this purpose. I wrote a simple GUI-O application example which I ported over to the Arduino Nano Matter/MGM240S MCU. The link to that is: https://www.gui-o.com/examples/gui-o-and-boards [2].

The MGM240S MCU is capable of transmitting BLE packets at a transmit power of 10dBm, which is comparable to the ESP32S3's maximum BLE Tx power of 9dBm. At this power level, the MGM240S draws 23.4mA, according to the datasheet. When I did current consumption measurements while running a BLE sketch, I measured a current draw of around 5.7mA which corresponds to a transmit power of 0dBm. This is the default Tx power setting, and I was unable to find a function to change this.

MY FIRST MATTER-ENABLED PROJECT

There are many commercial IoT products that work with Alexa, Google Home, etc. I wanted to do something more esoteric with either the Nano Matter or Sparkfun Thing Plus Matter boards. If you check out my earlier column in Circuit Cellar issue 412, you'll see that I built an IR-Blaster to turn my Fujitsu heat pump on/off, using Alexa commands.

For this column, I decided to design a different IR-Blaster. This time, it would be a universal IR-blaster: i.e. it would be able to "learn" IR codes from many different IR remotes (i.e. those that use carrier frequencies different than the 32KHz that the Fujitsu uses). Also, it would connect to the Alexa hub using the Matter protocol and not just emulate an IoT controlled power outlet, like the project described in issue 412. Lastly, it would be able to control more than device, if necessary.

I won't discuss the project's software in terms of the Matter connectivity, as I've covered that already. I'll just mention that I tailored the software around the Arduino example program "Matter lightbulb-multiple". Specifically, I define 2 "lightbulbs" in the program and include code to receive Matter messages for each of these two devices. There are only 2 commands per device: on and off. Therefore, there can be 4 discrete commands arriving, and each of these is associated with a specific IR code. When a command is received over the Matter protocol, the appropriate IR code is sent- using the IR LED contained in the project. For the following discussion, please refer to **Figure 4**, the device's schematic diagram.

The project's sketch includes an IR "learning" function. This mode is entered when switch SW1 is held down as the device is being powered up. I use a TSMP98000 IR receiver module. Unlike most IR receiver modules, this unit does not contain a bandpass filter, tuned to a specific IR carrier frequency. It is designed specifically for the learning function used in IR blasters and universal remotes. That is, it will respond to any IR remote which uses a carrier frequency in the 25-60KHz region. Because it does not contain bandpass filtering, it's more susceptible to interference, etc. Therefore, for learning purposes, you should place the IR remote that you are emulating, no further than a foot away from the TSMP98000 receiver module.

I have built projects that perform IR learning in the past using ESP32 MCUs. ESP32s have a dedicated IR remote function block (RMT), which handles the IR signal timing completely in hardware. The MGM240S does not contain anything like this RMT function block. However, I was able to handle the IR learning function, as well as IR code transmission, using only software and the micros() function.

The ESP32 contains two discrete processors: one to handle the RF functions and the other for the user's program. This is handy, as it means that the user program has 100% control of the execution time on its dedicated processor. Any randomly timed RF packets coming in are handled in real-time, by the second processor present in the ESP32.

The MGM240S MCU contains a separate Arm Cortex M0 MCU for its RF section. However, I noticed that when Matter commands were being received at the same time as an IR packet was being transmitted, the transmission of the IR packet was sometimes interrupted. Therefore, in my sketch, I delay the sending of the IR packet a few seconds after the Matter command has arrived. I also measure the time that it takes for the IR packet to be transmitted, and if it doesn't match the IR packet timing (measured during the learning process), I repeat the process until the timing does match. This guarantees that a valid IR packet has been transmitted.

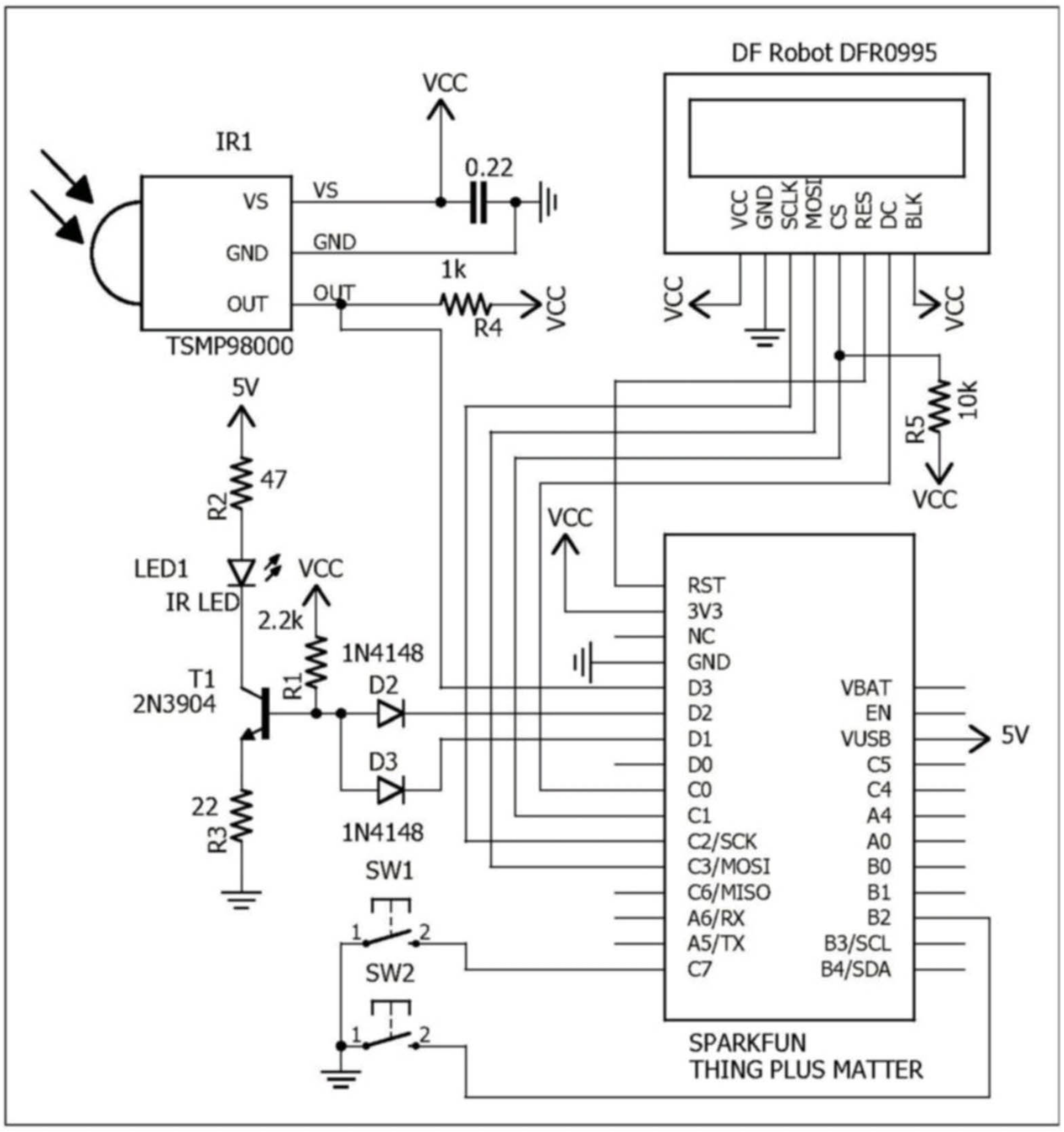


FIGURE 4
This is a schematic diagram of the Matter-controlled IR blaster project that I built.

The MGM240S MCU doesn't contain a RMT block to handle the IR carrier generation for transmission like the ESP32 does. However, I used the Arduino "tone" function to generate the carrier frequency. I verified that the MGM240S MCU, using the "tone" command, can generate carriers up to 60KHz, quite accurately. Referring to Figure 4, you can see diodes D2, D3 connected to port pins D2 and D1. The "tone" command outputs its square wave on the D2 pin and the carrier on/off gating comes from the D3 pin. The diodes act as an AND gate, in this configuration. The gated carrier signal drives a 2N3904 transistor, which in turn powers the IR LED. The IR LED is powered by the 5V supply, but the remaining circuitry is powered by 3.3V supplied by the LDO present on the Sparkfun Thing Plus Matter board.

Initially, I planned to store the IR codes in a μSD card, so I built the project using the Sparkfun Thing Plus Matter board, since it contained a μSD card socket. However, as I mentioned earlier, I was unable to get the DFR-0995 TFT display working when both the Matter protocol was loaded, and a μSD card was installed. That led me to pivot and investigate the EEPROM function. In other MCUs that I use, including the MGM240S, there is no actual EEPROM memory block present in the device. Instead, the MCU's flash memory is used to emulate EEPROM. It turns out that the MGM240S contains 10240 bytes of EEPROM memory. This is sufficient to store about 10 IR codes- more than needed for this project. Note however that the EEPROM function only works if you have the Matter protocol loaded with your sketch. This fooled me at first, and I cannot explain why this is the case. I also noticed that saving a 1kbyte EEPROM block took over 10 seconds, which is rather slow. It's not important in this application, but it might be meaningful in other applications.

In hindsight, had I known I was going to end up using the EEPROM storage and not the μSD card, I would have used the Arduino Nano Matter board for this project- saving the Sparkfun Thing Plus Matter board for a project that would have made use of the μSD card and/or the on-board LiPo charger.

Note that while this emulated EEPROM is not erased when a sketch is modified and re-

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] to [2] as marked in the article can be found there.

Resources

Arduino | www.arduino.cc

Sparkfun | www.sparkfun.com

loaded, it is erased when the Arduino IDE's "burn bootloader" command is run. Therefore, if you have to de-commission the device (to allow it to be re-discovered by Alexa), and you do so by burning the bootloader, the saved IR codes will be erased and will need to be re-learned.

- The functions performed by SW1 are as follows:
- Enter "learning" mode, when pressed at power-up
- Answer "yes" to the "save IR code" question (during the "learning" process)
- Send the first stored IR code (during normal operation)
- The functions performed by SW2 are:
- Answer "no" to the "save IR code" question (during the "learning" process)
- Send the second stored IR code (during normal operation)

In normal (non-learning) operation, the device will first attempt to join the Thread network. Assuming it can connect to the Alexa device, it will indicate on the TFT display that it has connected to the thread network- this takes only a few seconds generally. After that, it might take a little while before Alexa "discovers" that the device is online and acknowledges this. At this point the TFT display will indicate that the device is online with the Matter network.

Any subsequent commands coming in will be parsed and the applicable IR code sent out. The command number (0-3 in my existing sketch) will be displayed on the TFT display. The first two IR codes can also be sent out using the SW1, SW2 pushbuttons.

The Arduino Matter library was ported from the Silicon Labs library. It provides only a high-level access to the Matter protocol. For example, the matter_lightbulb library contains the following routines:

- set_onoff()
- get_onoff
- toggle()
- set_startup_onoff()

In the Nano Matter example programs, the get_onoff() routine is called periodically in the loop() function. It merely returns a true/false value which reflects the last valid Matter message it received. The reception of the Matter messages is done entirely as a background process: your MGM240S sketch has no way of knowing when such a message arrived. The Alexa hub, on the other hand, will receive an acknowledgement from your project's MGM240S that the message has been received. With that, it can also maintain the on/off status of the device being controlled by the MGM240S.

This system works fine unless your IR-controlled device is also being turned on/off using its own IR remote. In this scenario, neither the MGM240S in your project, nor the Alexa hub can keep track of the device's on/off status. In this case, the solution is as follows:

- If you are using the Alexa app and notice that its display of the status of the device is different than what you know to be the case, then press the on/off icon twice, to bring Alexa in sync with the device.
- If you are using a voice command, then just tell Alexa to do the opposite of what you want and then repeat the desired command.

The source code for this project can be found on *Circuit Cellar*'s website in the article support section.

Figure 5 shows the completed project. The 5V power for the unit is supplied by an external wall adapter.

CONCLUSIONS

When I received my Arduino Nano Matter board, as part of the Early Adopters promotion, I was unable to get it to work as anything but a Matter lightbulb. None of the other numerous examples worked, at least on my Alexa hub—so I put the board back on the shelf, where it collected dust for several months.

Luckily, the Arduino IDE presents a pop-up screen indicating when new versions of the various BSPs are available. The 2.2.0 version of the Silicon Labs BSP finally worked with my Alexa hub—I tested most of the example devices and they all worked. Table 4 shows the various example devices showing the ones which I tested out.

At this point, I can recommend you give either of the boards mentioned in this article a try, if you are interested in designing a Matter-enabled device and are familiar with the Arduino IDE. The Arduino-ported BSP and Matter library are much less complicated than the original Silicon Labs BSP and its Matter library. Also, it's strictly a personal bias, but I found the Silicon Labs Simplicity Studio IDE hard to understand/use when I first encountered it a few years ago.

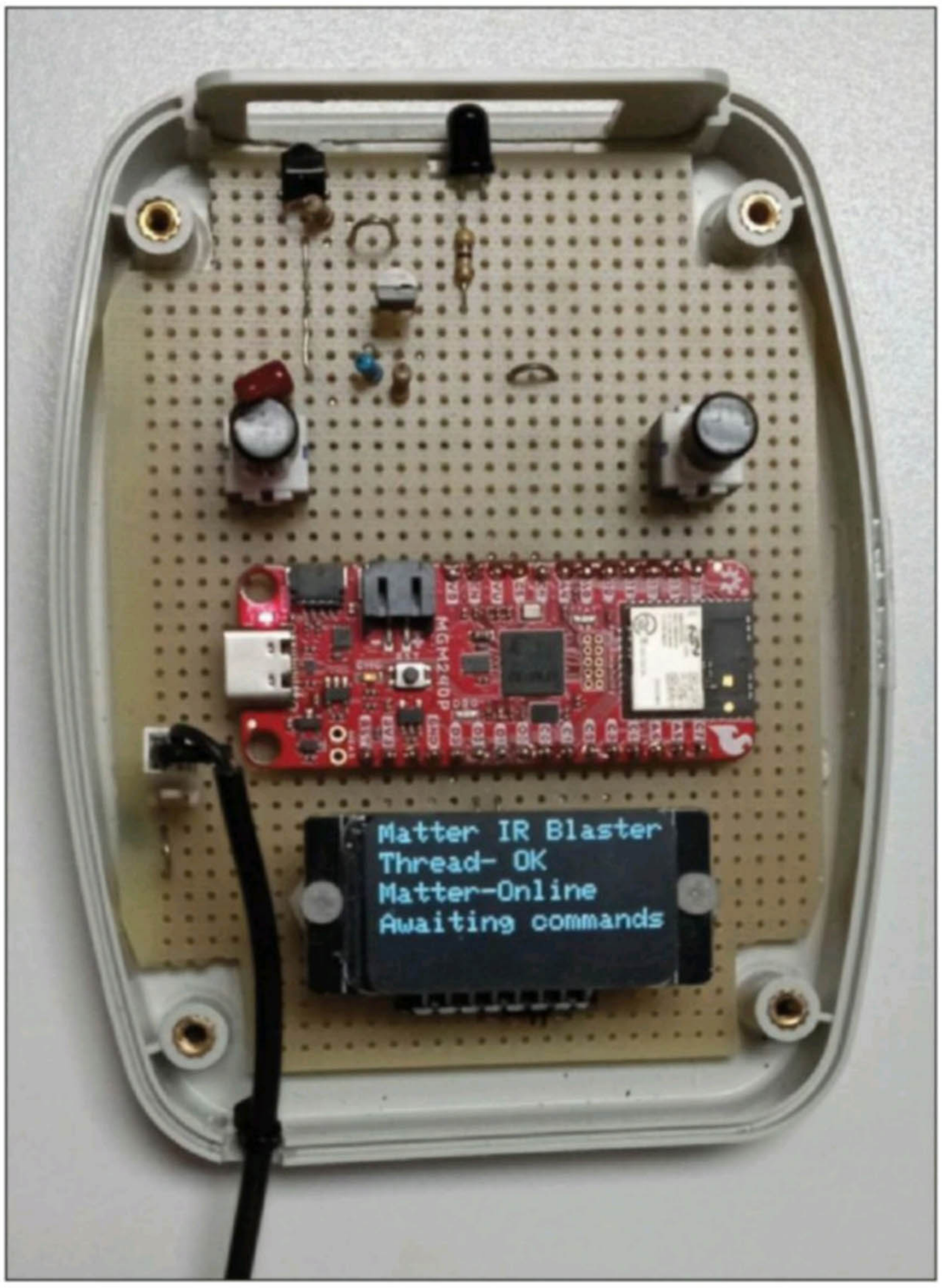


FIGURE 5
This is a photo of the finished project, less the top cover. The TSMP9800 IR receiver and IR Led are mounted just behind a cutout in the enclosure, at the top.

PRODUCT NEWS

By Kirsten Campbell

Trident IoT Software Development Kits For Z-Wave/ZWLR And Zigbee RF SoCs

Trident IoT, is proud to announce the release of its highly anticipated software development kits (SDKs) for its latest line of Z-Wave/ZWLR and Zigbee RF system-on-chips (SoCs). These cutting-edge SDKs are designed by developers, for developers, to streamline and accelerate the creation of innovative IoT solutions.

With a strong commitment to empowering product development, Trident IoT's new SDKs provide an intuitive and efficient platform for building, testing, and deploying IoT applications. At the core of device development is ELCap, Trident IoT's powerful command-line development tool, enabling seamless management of Trident SDKs. ELCap offers a user-friendly interface for developing, building, and flashing Trident SoC-based projects, all from a single, cohesive application.

For gateways, Z-Way Controller SDK by Trident is the solution. Multi-protocol, multi-OS development kit ranging from C-Libraries up to a full stack implementation.

"Our goal is to make IoT development easier, faster, and more predictable," said Mariusz Malkowski, CTO at Trident IoT. "By offering a robust and developer-friendly SDK, we are enabling our customers to bring superior IoT products to market with increased efficiency and reduced complexity."

Whether for smart home devices, gateways, or other advanced wireless IoT applications, Trident IoT's SDKs deliver unparalleled performance and ease of use. With support for multiple operating systems developers can now build scalable, high-performance solutions with confidence.

Trident IoT | tridentiot.com

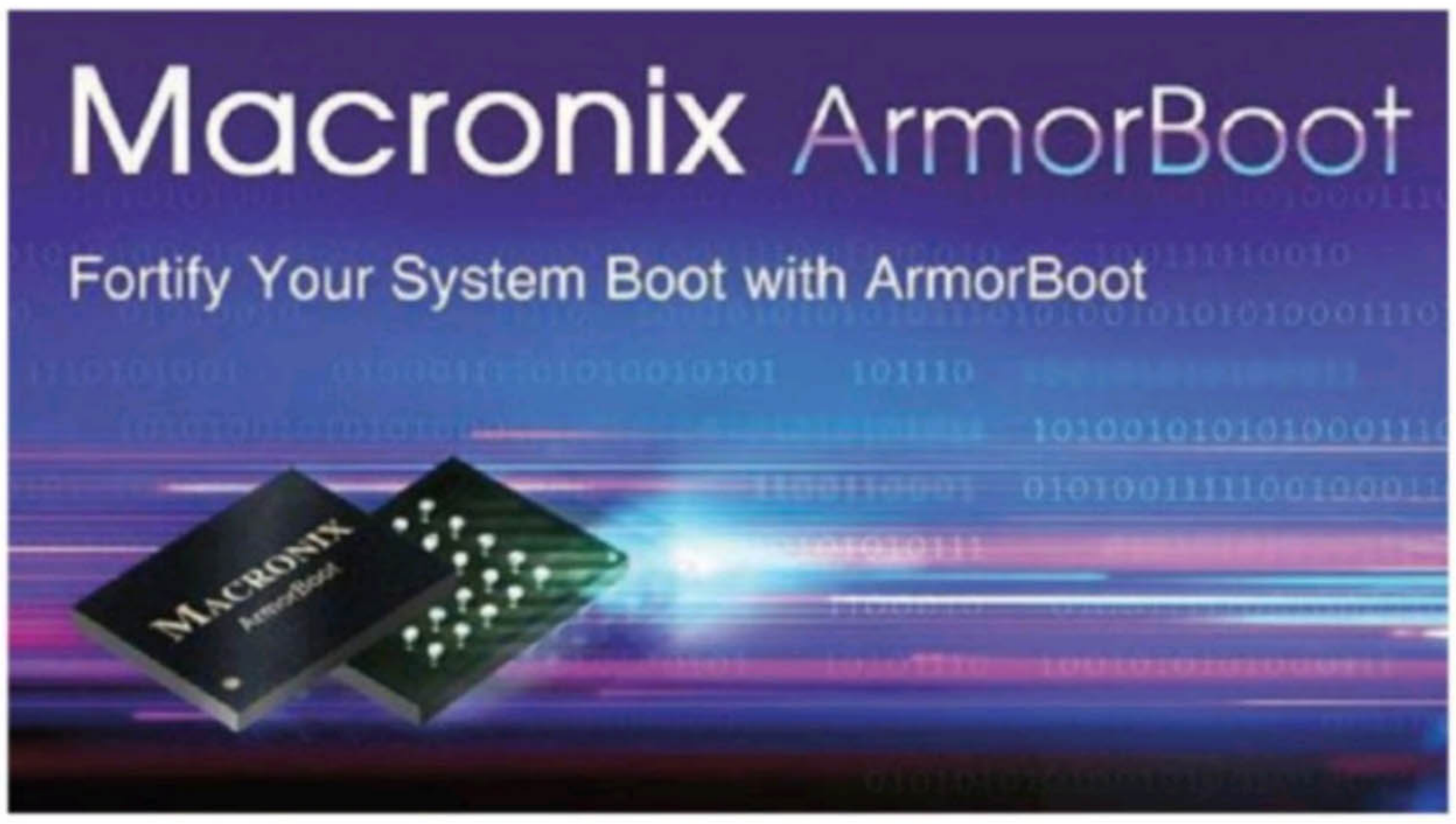
Wi-SUN
45°
Trident IoT
Z-Wave
LoRa
Matter

Macronix Introduces ArmorBoot MX76 for Security and High Performance in Automotive IoT Markets

Macronix, a leading global non-volatile memory (NVM) integrated device manufacturer, has launched its all-new ArmorBoot MX76 series flash memory. The ArmorBoot MX76 is designed for rapid initialization of electronic devices in diverse application scenarios. Authentication, data protection, and configuration are all completed simultaneously to provide Macronix customers with the winning edge in very challenging AI environments.

ArmorBoot is designed to speed up boot times. It is equipped with the standard SPI interface and supports both 1.8V and 3V for optimal power efficiency. The integration of advanced encryption algorithms, authentication, and data integrity verification technologies ensures data security at every stage of system operation. In addition to rapid authentication when a device is started for the first time, it also protects critical data against external interference during system upgrades and updates.

ArmorBoot MX76: Providing the ultra-fast startup time the market has been clamoring for. Device boot times and data protection during the update process now play a crucial part in protecting system stability and user experience. The product's high-speed boot capability is combined with encryption technology and a secure boot mechanism to complete the authentication and data verification processes several times faster than legacy solutions.



The ArmorBoot MX76 owes its outstanding performance during system updates to its enhanced anti-replay attack mechanism. Protection against unauthorized attacks during version updates or execution of secure commands by the system effectively prevents data tampering, malicious attacks, and virus infections. If the system is subjected to security threats or anomalous situations, the integrated advanced protection mechanism switches rapidly over to a secure mode to maintain system stability and greatly reduce security risks. Whether or not used for initial startup or system upgrades/updates, ArmorBoot MX76 works quickly and effectively to protect data against external threats.

Smart integration for high-performance and seamless upgrades, ArmorBoot MX76 features not only advanced data security technology but also enhanced system compatibility.

ArmorBoot MX76 is based on Macronix's advanced production process to deliver consistent product quality and high performance. Exceptional production yield and operating performance make this a solution that combines both security and performance, as well as the ideal choice for equipment upgrades or integration of existing systems. Available capacities range from 256 MB to 1 GB.

As the pace of digital transformation continues to speed up, Macronix is not only engaging in continued innovation in memory technology but also paying close attention to market requirements on device startup and update security.

Macronix | https://www.macronix.com

Direct Insight Creates QNX 8 BSP for QS93 i.MX93 Solder-Down SoM

Direct Insight, the UK-based technical systems integrator and reseller of system-on-module (SoM) and other embedded systems, has created a QNX 8 BSP (board support package) for Ka-Ro's QS93 solder-down module with NXP's i.MX93 processor. The combination of a feature-rich real-time OS allied to a powerful processor with dual ARM Cortex-A55 cores suits high-reliability IoT and embedded systems applications including industrial automation, smart homes and cities, building control and energy, automotive and mass transportation, medical devices, robotics, and more—including safety-critical applications requiring IEC 61508 or IEC 62304 certification.

Ka-Ro Electronics' QS93 is a 27mm x 27mm QFN-style solder-down SoM featuring NXP's new i.MX93 low power, high performance processor. Dual ARM Cortex-A55 cores running at 1.7GHz deliver performance comparable with earlier quad-core processors. The energy-efficient processor uses less than 1W under normal load conditions—and its optional low-drive mode can reduce that to sub-500mW. An industrial operating temperature range of -25°C to +85°C is supported, and the solder-down SoM format offers low cost and provides both excellent EMI performance and highly efficient thermal characteristics, meaning that usually no heatsink is required—ideal for space-constrained applications.

QNX board support packages (BSPs) provide an abstraction layer of hardware-specific software that facilitates implementing QNX software in your system. Architecture- and board-specific, BSPs look after initializations and other hardware-specific tasks needed to prepare an environment in which your embedded system can run, serving as a delivery mechanism for the components that define startup behaviors. David Pashley, founder and Managing Director at Direct Insight comments, "Our in-house developed BSP unlocks the full benefits of the QNX OS for the power-efficient Ka-Ro QS93 modules. QNX is ideal for real-time mission-critical, cybersecurity-conscious and safety-certified applications—and the BSP created by Direct Insight provides an ideal starting point for designing secure, high-availability industrial or medical embedded systems, lowering project risk and shortening time-to-market."

Direct Insight | directinsight.co.uk

GigaDevice's Dual-Power Supply SPI NOR Flash for 1.2V SoCs Halves Read Power Consumption

GigaDevice, a leading semiconductor company specializing in flash Memory, 32-bit microcontrollers (MCUs), sensors, analog products and solutions, has unveiled the GD25NE series of dual-power supply SPI NOR Flash, designed specifically for 1.2V system-on-chip (SoC) applications.

The GD25NE series strengthens GigaDevice's strategic dual-power supply Flash roadmap, providing seamless compatibility with next-generation 1.2V SoCs and eliminating the need for an external booster circuit. With its higher performance and lower power consumption, the GD25NE series addresses the growing demand for advanced embedded storage, making it an ideal choice for wearable devices, healthcare, IoT, data center and Edge AI applications.

Combining a 1.8V core voltage with a 1.2V I/O interface voltage, GD25NE supports single, dual, quad single transfer rate (STR) and quad double transfer rate (DTR) operation. It achieves high-speed clock frequencies of up to 133MHz in STR mode and 104MHz in DTR mode.

With a typical page program time of 0.15ms and sector erase time of 30ms, the GD25NE series significantly outperforms conventional 1.2V-only Flash solutions—offering up to 20% faster read performance, over 60% faster program speed, and 30% reduction in erase time.

The GD25NE series is engineered with ultra-low power consumption by design, making it ideal for energy-sensitive applications. It features a typical deep power-down current of just 0.2µA, a Quad I/O DTR read current of 9mA at 104MHz, and program/erase currents as low as 8mA. Compared to conventional 1.8V solutions, the 1.2V design reduces power consumption by up to 50%.

"The GD25NE series represents a new class of dual-supply SPI NOR Flash, delivering an optimal balance of high performance and ultra-low power consumption," said Ruwei Su, GigaDevice vice president and general manager of Flash BU, "With significantly reduced power usage, faster read speeds, and enhanced program/erase efficiency, this solution is designed to meet the evolving demands of next-generation 1.2V SoCs. As part of our ongoing commitment to innovation, we continue to expand our portfolio, providing customers with more efficient, reliable, and future-ready Flash solutions for new leading-edge applications."

The GD25NE series offers various densities ranging from 32Mb to 256Mb, and supports SOP16 (300 mil) and BGA24 (5×5 ball array) packaging options. The first product in the series, the 256Mb GD25NE256H is available in samples. The rest of the product series from 32Mb to 128Mb density will follow.

GigaDevice | gigadevice.com

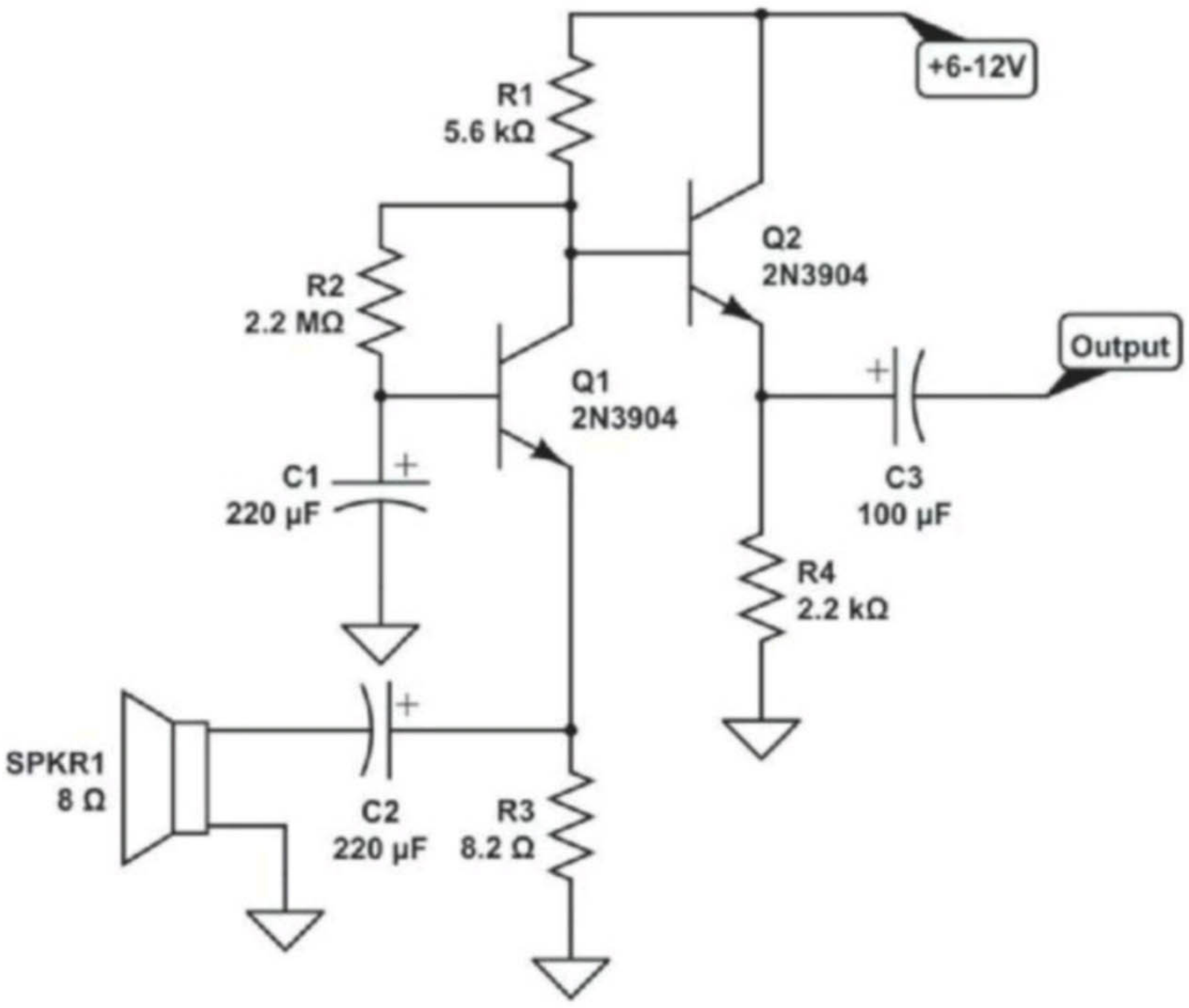
GigaDevice
GD25NE series
New Dual Power SPI NOR Flash
for Next Gen 1.2V SoCs
GigaDevice
GD25NE SPI NOR Flash
High-speed read
for half the power

TEST YOUR EQ

Contributed by David Tweed

Question 1 —

The following circuit was recommended as a preamp for a speaker being used as a microphone in an intercom system.


R1
5.6 kΩ
+6-12V
Q2
2N3904
R2
2.2 MΩ
Q1
2N3904
Output
C1
220 µF
C3
100 µF
R4
2.2 kΩ
SPKR1
8 Ω
C2
220 µF
R3
8.2 Ω

What is the configuration of Q1 called?

Question 2 —

What are the advantages of using this circuit configuration for this application?

Question 3 —

To what extent does the nonlinearity of the transistor's B-E junction contribute to distortion of the signal at the collector?

Question 4 —

What are other disadvantages of this circuit?

POWERING INNOVATION:
DISCOVER THE R24C2T25 –
THE NEXT GENERATION IN
DC/DC CONVERSION

SEAMLESSLY INTEGRATE CUTTING-EDGE EFFICIENCY WITH OUR 2W
ISOLATED SMD CONVERTER FOR ADVANCED IGBT/SIC GATE DRIVERS

RECOM
recom-power.com/gate-drivers

RECOM
R24C2T25 series
12.83 mm
7.5 mm
3.18 mm

IDEA BOX

The Directory of PRODUCTS & SERVICES

AD FORMAT:
Advertisers must furnish digital files that meet our specifications (www.circuitcellar.com/mediakit).

All text and other elements MUST fit within a 2" x 3" format.
E-mail adcopy@circuitcellar.com with your file.

For current rates, deadlines, and more information contact
Hugh Heinsohn at 757-525-3677 or Hugh@circuitcellar.com

ADuC841
Microcontroller
Design Manual

ADuC841 Microcontroller Design Manual

From Microcontroller Theory to Design Projects

If you've ever wanted to design and program with the ADuC841 microcontroller, or other microcontrollers in the 8051 family, this is the book for you. With introductory and advanced labs, you'll soon master the many ways to use a microcontroller. Perfect for academics!

Buy it today!
cc-webshop.com

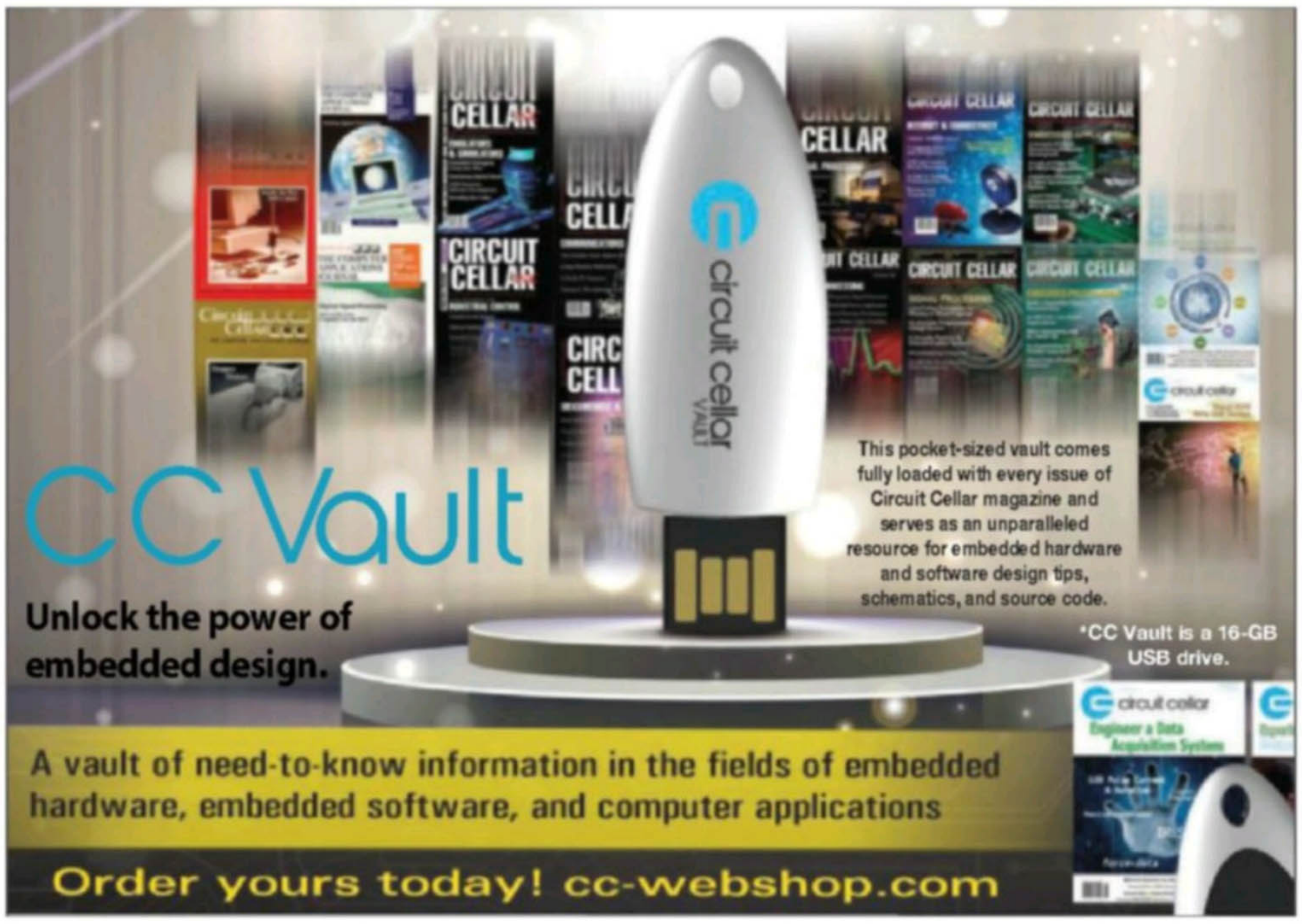
RAPID18
Universal Development
Kit for PIC18 Family

- Real-time Clock/Calendar with Supercap
- USB port for text communication to the running program
- Built-in bootloader for loading code
- Excellent for AC/DC Data Logging

Kit includes everything you need to develop with a PIC® MCU:
- Rapid 18 Prototyping Board
- Single-Chip IDE C Compiler
- Exercise Book
- Power Adapters and Cables

sales@ccsinfo.com (262) 522-6500 x35
www.ccsinfo.com/cc1223

PIC® MCU and dsPIC® are registered trademarks of Microchip Technology Inc.



CC Vault

Unlock the power of embedded design.

This pocket-sized vault comes fully loaded with every issue of Circuit Cellar magazine and serves as an unparalleled resource for embedded hardware and software design tips, schematics, and source code.

*CC Vault is a 16-GB USB drive.

A vault of need-to-know information in the fields of embedded hardware, embedded software, and computer applications

Order yours today! cc-webshop.com

embeddedTS
embeddedTS.com

TS-7100-Z

Powered by the high performance, ultra-efficient Arm® Cortex®-A7 based i.MX6UL CPU

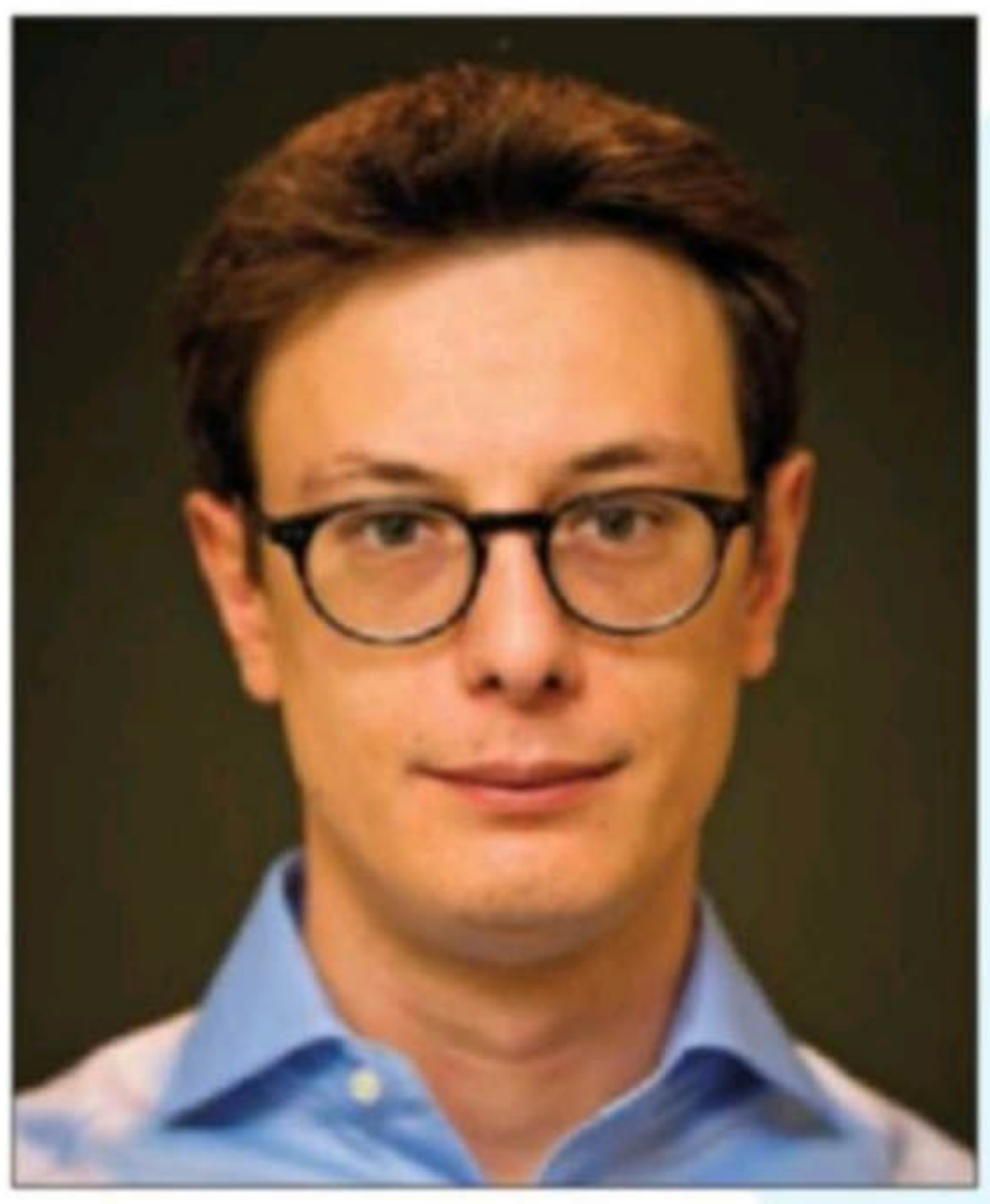
...In an enclosure as small as two decks of cards
(2.4" x 3.6" x 1.7")

Best In Show
Embedded Computer Design

Made in the USA
with global parts

The Future of Cryptography

PQCryptoLib-Embedded and the Quantum Threat



By
Michele Sartori
Senior Product Manager at PQShield

The invention of quantum computing could be another significant leap in technology, projecting us forward into a new era of growth and progress. It also presents a serious cybersecurity threat to embedded systems everywhere.

A quantum computer leverages the principles of quantum mechanics to generate enormous computational power that classical supercomputers are unable to do. As a result, they could be able to solve specific problems, such as complex modeling required for drug discovery or the creation of energy-efficient systems and processes.

However, quantum computers could also be used to break a very different kind of problem: the mathematical techniques that currently keep our online data safe in traditional Public Key Cryptography.

Public Key Cryptography algorithms rely on mathematical "trapdoors," operations that are easy to apply, but difficult to undo without additional information. The central idea is that a public key lets you apply the trap door (encrypt), and only the corresponding private key enables you to safely decrypt. Without the private key, the information remains secure because the mathematical problem is too difficult to solve.

In the 1990s, mathematicians began developing new techniques to figure out how to solve some of these "hard" problems. Peter Shor's algorithms were major breakthroughs, showing for the first time that there is a path to breaking public-key cryptography. To run Shor's algorithms, a computer would require a vast amount of processing power—far more than is possible for any classical computer.

Now, the rapid advancements in quantum computing mean the public key schemes that protect our embedded systems today, such as RSA and ECC, are at risk of being broken soon.

NIST STANDARDIZATION

As a result of the quantum threat, the field of post-quantum cryptography (PQC) has emerged—a community of academics, researchers, engineers, and organizations, with a view to updating cryptography to quantum resistance.

In 2016, the United States National Institute of Standards and Technology (NIST), launched the formal process to develop and standardize a set of post-quantum algorithms, submitted by the post-quantum cryptography community.

The aim was to evaluate various PQC candidates for two primitives:

- ***Key Agreement*** (which includes key exchange, public key encryption and key encapsulation mechanisms, or KEMs).

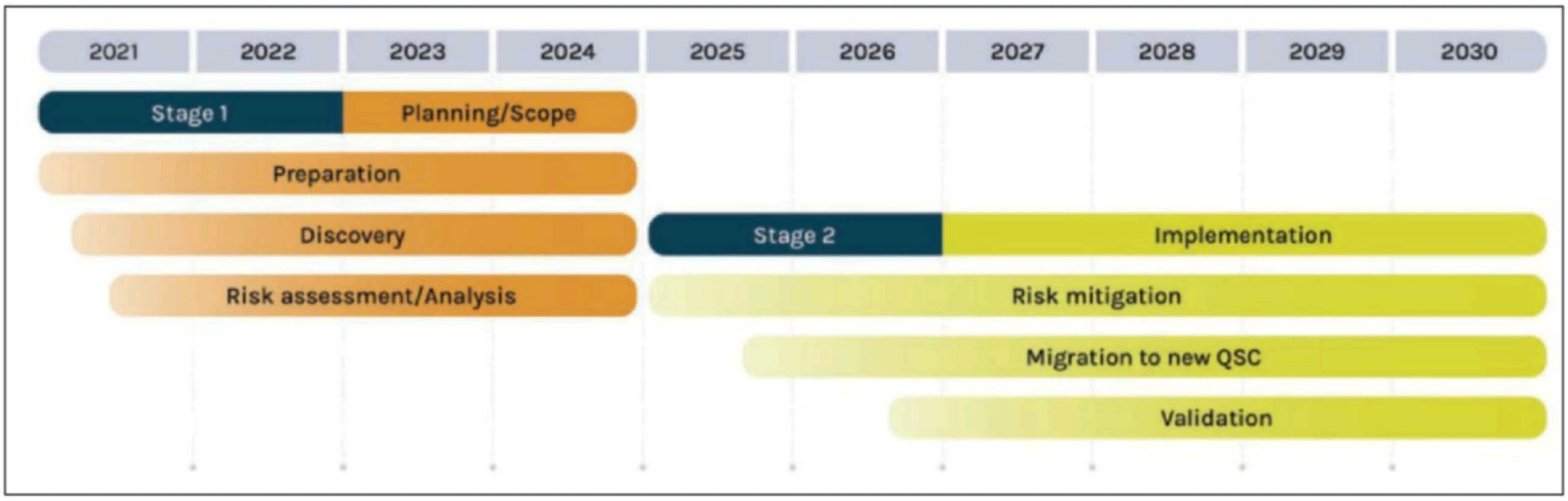


FIGURE 1
Example timeline for PQC preparation and migration from the Canadian National Quantum-Readiness Working Group.

- ***Digital Signatures*** (providing authentication, integrity, and non-repudiation of a digital message).

In 2024, after three rounds of candidates, NIST announced that the following three post-quantum algorithms would be standardized as FIPS 203, FIPS 204, and FIPS 205.

- ***FIPS 203 ML-KEM*** (Kyber) Module-Lattice-Based Key-Encapsulation Mechanism Standard
- ***FIPS 204 ML-DSA*** (Dilithium) Module-Lattice-Based Digital Signature Standard
- ***FIPS 205 SLH-DSA*** (SPHINCS+) Stateless Hash-Based Digital Signature Standard

One additional algorithm is also expected to be published in due course.

- ***FN-DSA*** (FALCON) Fast Nominative Digital Signature Algorithm

The publication of standards is a milestone in the story of PQC, and it's certainly come at a point of transition. **Figure 1**, an example timeline recommended by the Canadian National Quantum-Readiness Working Group, shows how the shift from Stage 1 (preparation for migration) is impacting the conversation, shifting the focus to Stage 2, implementation and compliance.

PROTECTING EMBEDDED SYSTEMS

As NIST's standards come into place, bring embedded systems into alignment with them presents a specific challenge.

Embedded systems are often specialized by design and support limited resources such as memory and bandwidth. They can have long shelf-lives and be difficult to replace, and it's possible that a device is susceptible to physical vulnerability, being remote or deployed in an accessible location.

As a result of these factors, embedded devices and systems can be particularly susceptible to threats, and efficiently upgrading their cryptographic protections might be more difficult than with general-purpose systems.

As industries move further toward PQC-compliance, the necessity to upgrade embedded systems increase. What migration strategies are already available to designers and manufacturers?

THE REGULATORY LANDSCAPE

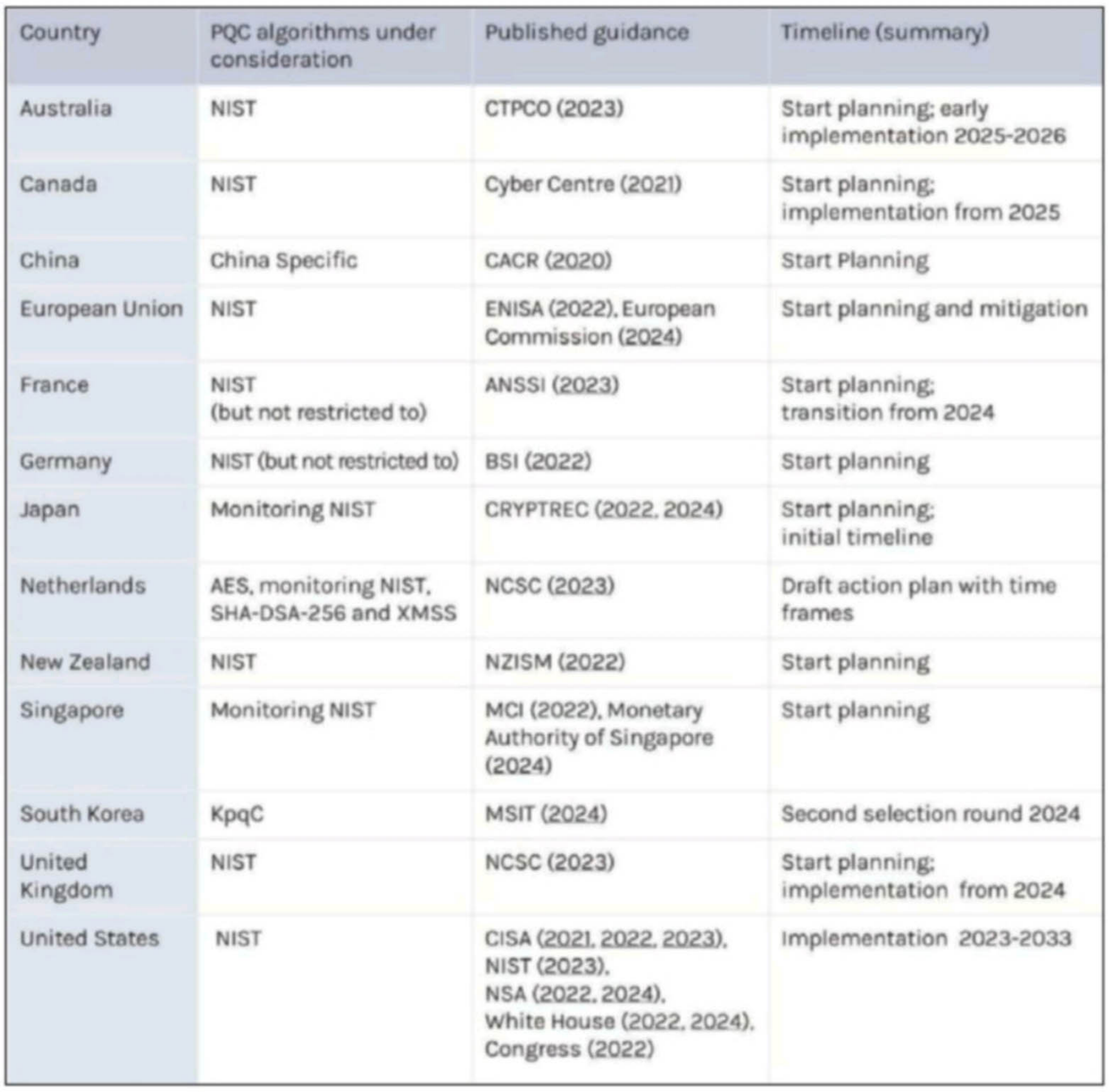
The last few years have seen security bodies and agencies around the world take notice of the need to implement PQC, with the US government releasing key requirements for federal agencies (such as CSNA 2.0) and organizations such as the German Federal Office for Information Security (BSI), the UK's National Cyber Security Centre and the French national security agency (ANSSI) releasing key guidance and timelines for migration (**Figure 2**).

The one consistent message across all the published guidance is clear, organizations should already be planning the transition to post-quantum cryptography, and for many governments, the time to begin implementation of PQC algorithms is now.

How can this be applied practically to embedded systems?

PQCRYPTOLIB-EMBEDDED

As the regulatory landscape and NIST standardization efforts highlight, protecting embedded systems against quantum threats is an immediate concern for industries reliant on secure, long-lasting devices. PQCryptoLib-Embedded is PQShield's response to this challenge, a state-of-the-art solution engineered to bring robust post-quantum cryptographic capabilities to memory-constrained environments. Designed to serve embedded systems with specific performance and resource limitations, PQCryptoLib-Embedded is compatible with a wide range of embedded devices, including microcontrollers and architectures such



Country	PQC algorithms under consideration	Published guidance	Timeline (summary)
Australia	NIST	CTPCO (2023)	Start planning; early implementation 2025-2026
Canada	NIST	Cyber Centre (2021)	Start planning; implementation from 2025
China	China Specific	CACR (2020)	Start Planning
European Union	NIST	ENISA (2022), European Commission (2024)	Start planning and mitigation
France	NIST (but not restricted to)	ANSSI (2023)	Start planning; transition from 2024
Germany	NIST (but not restricted to)	BSI (2022)	Start planning
Japan	Monitoring NIST	CRYPTREC (2022, 2024)	Start planning; initial timeline
Netherlands	AES, monitoring NIST, SHA-DSA-256 and XMSS	NCSC (2023)	Draft action plan with time frames
New Zealand	NIST	NZISM (2022)	Start planning
Singapore	Monitoring NIST	MCI (2022), Monetary Authority of Singapore (2024)	Start planning
South Korea	KpqC	MSIT (2024)	Second selection round 2024
United Kingdom	NIST	NCSC (2023)	Start planning; implementation from 2024
United States	NIST	CISA (2021, 2022, 2023), NIST (2023), NSA (2022, 2024), White House (2022, 2024), Congress (2022)	Implementation 2023-2033

FIGURE 2
Guidance and timelines for PQC migration published by global regulators and security agencies.

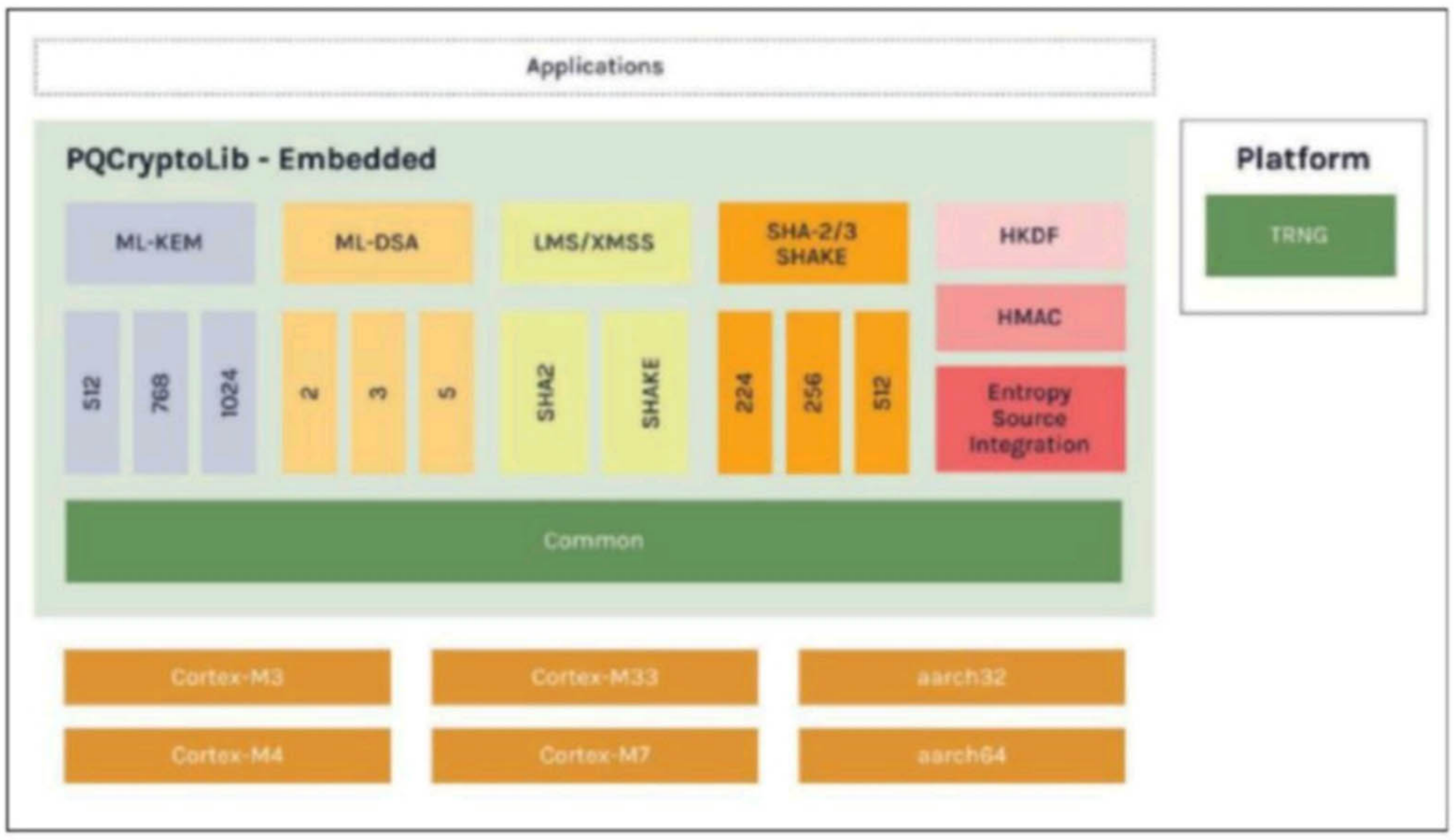


FIGURE 3
PQCryptoLib – Embedded: PQShield's post-quantum cryptography library for memory-constrained platforms.

as ARM, RISC-V, and x86. It integrates key post-quantum algorithms such as ML-KEM, ML-DSA, and SLH-DSA, alongside other non-standard post-quantum and traditional cryptographic algorithms. This robust combination enables PQCryptoLib-Embedded to support encapsulation mechanisms, signature schemes, as well as symmetric and asymmetric encryption (**Figure 3**).

With its highly configurable architecture, PQCryptoLib-Embedded enables developers to select only the cryptographic functions they need, minimizing binary size and optimizing memory use—an essential strategy for devices with limited resources. This flexibility aligns with international cryptographic standards such as FIPS 140-3, covering FIPS 202, 203, and 204, to meet stringent security requirements across global markets and achieving FIPS ACVP and CMVP certifications, while respecting ISO standards for implementation.

In anticipation of potential vulnerabilities, PQCryptoLib-Embedded includes built-in defenses against attacks that exploit malformed inputs, offering a line of defense against adversarial manipulation. Additionally, the libraries are subjected to rigorous fuzz testing, a process that significantly reduces the probability of implementation bugs, further enhancing system resilience. PQCryptoLib-Embedded is also designed to be resistant to timing-based cryptanalysis, ensuring uniform execution times that avoid timing vulnerabilities. Moreover, the ML-KEM implementation within PQCryptoLib-Embedded undergoes formal verification on selected platforms, providing mathematically proven assurances of security.

Optimized for bare-metal environments, PQCryptoLib-Embedded supports deployment on hardware directly in bare metal, which maximizes processing efficiency and reduces latency. This design is enhanced by the library's ability to integrate with hardware hash engines, facilitating the offloading of hash calculations to dedicated hardware and boosting performance for hash-based signature schemes like LMS, XMSS, and SLH-DSA.

By including both quantum-resistant and classical cryptographic support, PQCryptoLib-Embedded offers an adaptable pathway toward secure, quantum-resistant infrastructure. For industries that must balance security compliance with performance and memory constraints, PQCryptoLib-Embedded provides a foundation for long-term resilience, addressing the pressing needs raised by new regulations and standards.

BUILD SECURELY

Quantum computing presents a fundamental risk to our traditional cryptographic methods. In every single industry, the cryptography that keeps data, devices, connections, and components secure, needs to be modernized in order to align with the new global standards. This modernization needs to begin today.

As businesses embargo on their journey to quantum security, they must remember to think openly and build securely. Through products like PQCryptoLib-Embedded, organizations can adopt quantum-resistant, FIPS-ready cryptographic solutions to update the legacy cryptography components of the world's technology supply chain, and staying one step ahead of the attackers.

ABOUT THE AUTHOR

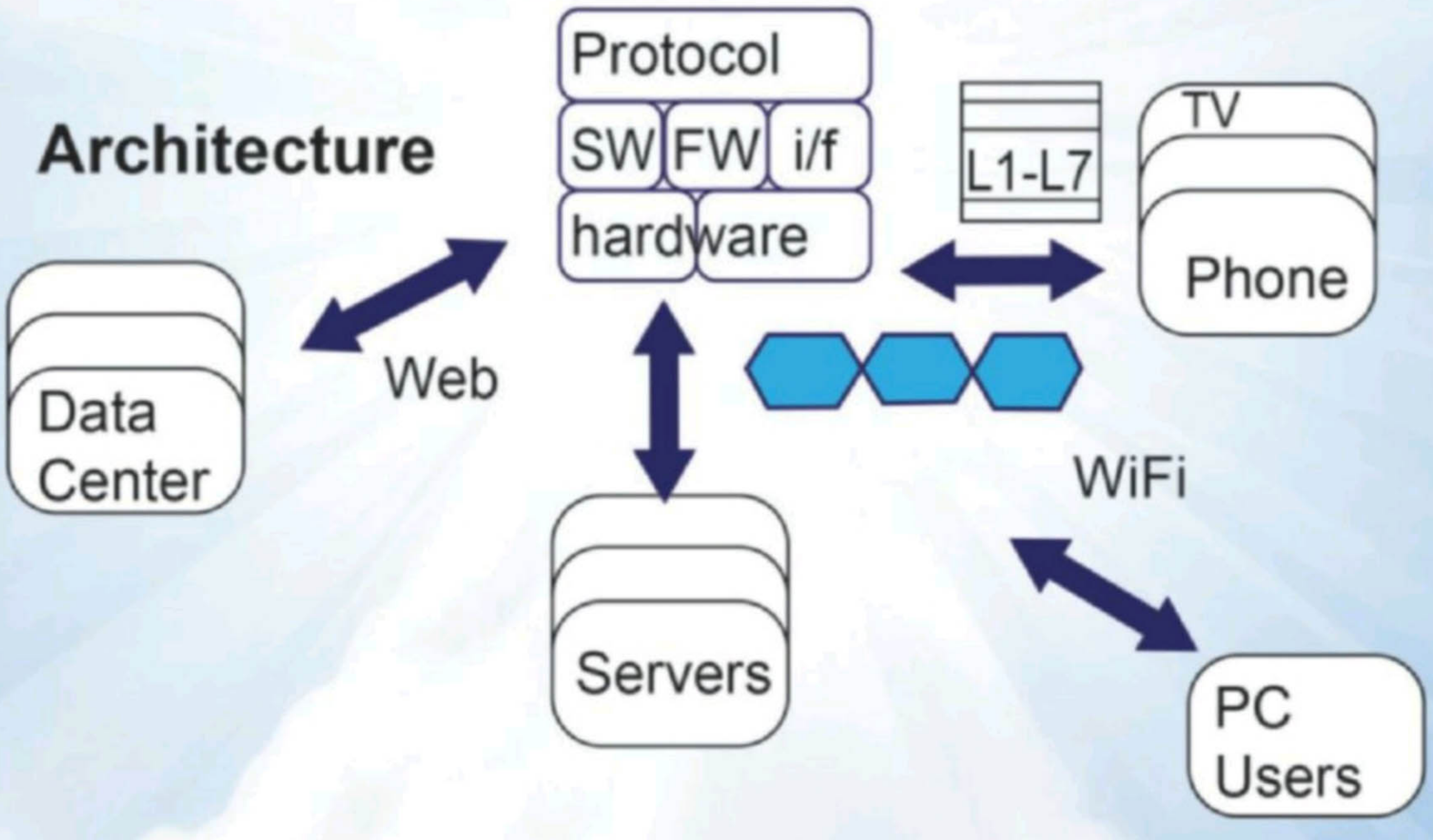
Michele Sartori is Senior Product Manager at PQShield, a post-quantum cryptography (PQC) company creating the global standards and core technologies to power the future security layer of the world's leading organizations. It is the only cybersecurity company that can deliver high-quality secure implementations of quantum-safe cryptography on chips, in applications, and in the cloud, and is also an authority on PQC side channel attack resistance, having built a dedicated PQC SCA test lab with industry partners. PQShield is also a leading contributor to the post-quantum cryptography standardization projects throughout the world.

RESOURCES

www.PQShield.com

SeaFire® Micros, Inc.

IP & Technology for Cloud Networks


Architecture
Protocol
SW
FW
i/f
hardware
L1-L7
TV
Phone
Data Center
Web
WiFi
Servers
PC Users

Design through Production
Military — Commercial

Hardware (Analog/Digital)
Firmware • Software • FPGA

Patented Network Technology | Unique Cloud Technology
No Compete Government Procurement
No Compete Government Funding

20% DISCOUNT on 1st Purchase

info@seafire.com | seafire.com

TS-9370

Low Power, High Performance SBC

The TS-9370 is a versatile and power-efficient industrial SBC built for rugged applications. Powered by the NXP i.MX9352 64-bit ARM® architecture plus two Cortex®-A53 processors operating up to 1.7 GHz, it delivers robust performance with low power consumption.

32 GB eMMC (upgradable)

2 GByte LPDDR4 RAM

MikroBUS socket
For MikroE Click Boards

Made in the USA
with global parts

embeddedTS.com
TS-9370
Rev.P3

VISIT US THIS MAY
IN HOUSTON, TX AT
OTC 2025
BOOTH #1675
XPONENTIAL
BOOTH #3725

embeddedTS

TS-4100

Ultra-Efficient and Fully Featured

The TS-4100 features an NXP i.MX 6UltraLite CPU, industrial temp range, FPGA, eMMC, and customizable I/O. Comes preloaded with Debian Stretch.

TS-7100-Z

As Small as Two Decks of Cards

Powered by an Arm® Cortex®-A7 i.MX6ul CPU, it runs efficiently at <2W, featuring Dual Ethernet, Bluetooth, WiFi, and NimbeLink/Digi sockets.

TS-7100-Z
embeddedTS.com
embeddedTS

www.embeddedTS.com